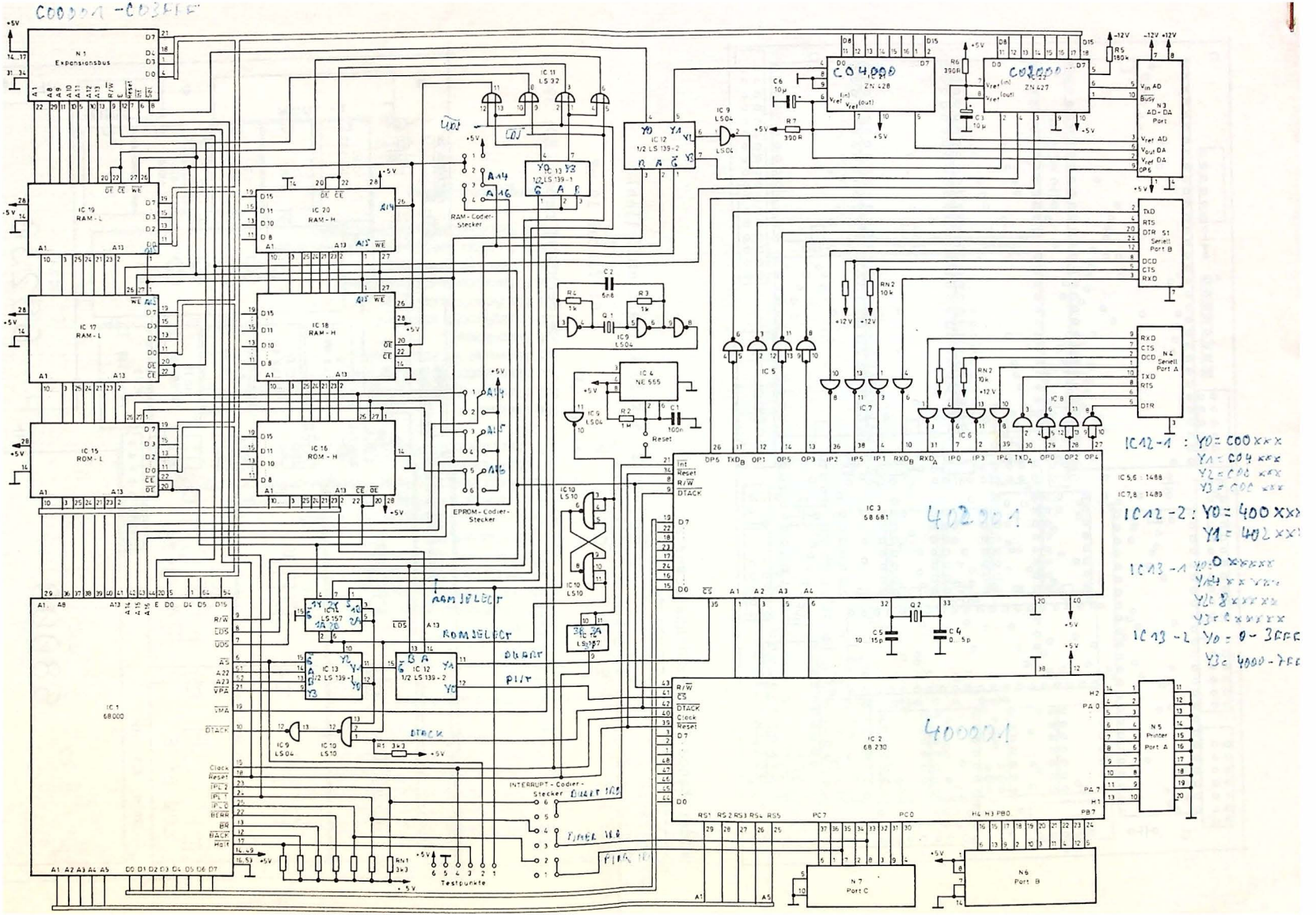
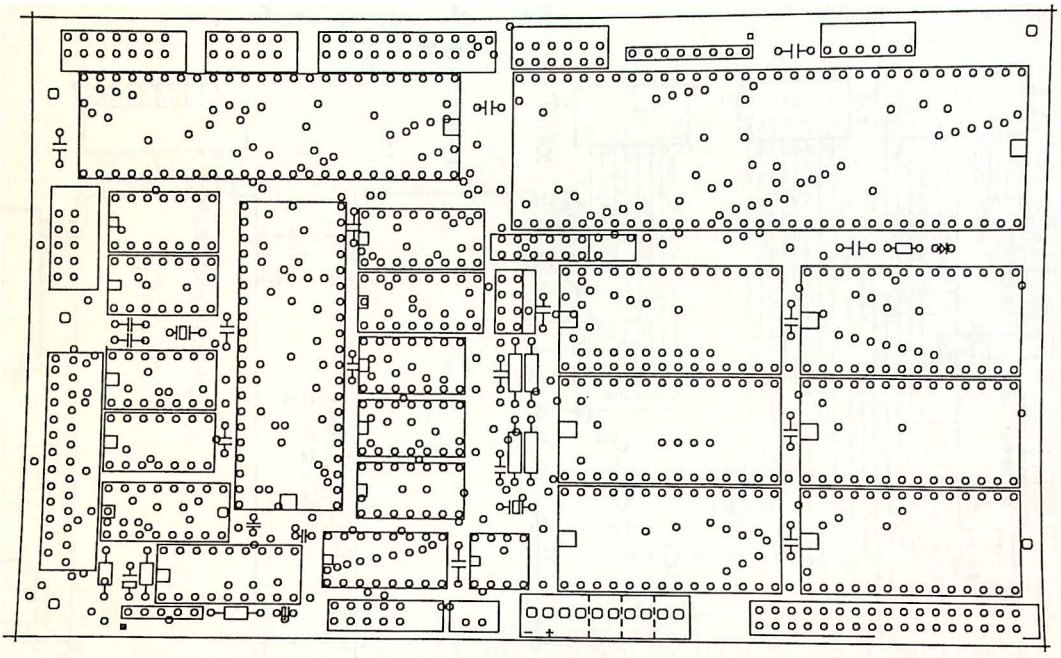
C'T KAT-Ce

68000 Einplatinencomputer

K leines
A allround
T alent
-
C omputer
e ffektiv

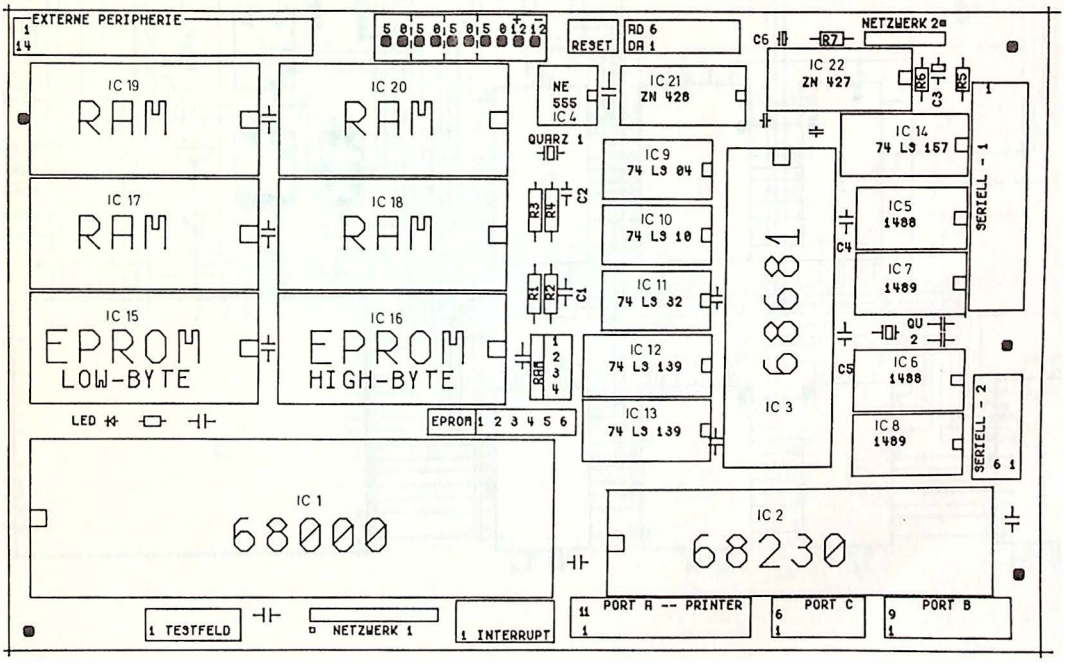

C00000 - C03FFF
IC 1 68000
IC 2 68 230
IC 3 58 681
IC 4 NE 555
IC 5 IC 6 IC 7 IC 8
IC 9 LS04
IC 10 LS10
IC 11 LS 32
IC 12 1/2 LS 139-2
IC 13 1/2 LS 139-1
IC 15 ROM - L
IC 16 ROM - H
IC 17 RAM - L
IC 18 RAM - H
IC 19 RAM - L
IC 20 RAM - H
RAM - Codier-Stecker
EPROM - Codier-Stecker
ROMSELECT
RAMSELECT
DTACK
INTERRUPT - Codier-Stecker
Testpunkte
N 1 Expansionsbus
N 2 Serell Port B
N 3 AD - DA Port
N 4 Serell Port A
N 5 Printer Port A
N 6 Port B
N 7 Port C
IC12-1 : Y0 = C00xxx
Y1 = C04xxx
Y2 = C08xxx
Y3 = C0Axxx
IC12-2 : Y0 = 400xxx
Y1 = 402xxx
IC13-1 Y0 = 0xxxxx
Y1 = 2xxxxx
Y2 = 4xxxxx
Y3 = 6xxxxx
IC13-2 Y0 = 0 - 3FFF
Y3 = 4000 - 7FFF



Widerstands- und Kondensatorwerte:

Netzwerk1: 7 x 3 k 3		Netzwerk2: 4 x 10 k (kann entfallen)	
R 1: 3 k 3	R 4: 1 k	C 1: 100 n	C 5: 10 p
R 2: 1 M	R 5: 390	C 2: 6 n 8	C 6: Tantal 10 uF
R 3: 1 k	R 6: 180 k	C 3: Tantal 10 uF	
R 4: 1 k	R 7: 390	C 4: 10 p	Widerstand neben LED: 390

alle anderen Kondensatoren: 100 n



Inhalt

1 Die C'T KAT-Ce ... ein paar Worte zum Kennenlernen 2

Inbetriebnahme, die ersten Schritte
2.1 Spannungsversorgung und Anschluß der KAT-Ce 3
2.2 Verbindung mit dem APPLE II+ 4
2.3 Verbindung mit dem COMMODORE C64 6
2.4 Verbindung mit dem GEPARD 6
2.5 Verbindung mit CPM-Rechnern 7

3 Aufteilung des Adressraumes 7

Die Software
4.1 Das Hauptmenü 8
4.2 Der Monitor 9
4.2.1 Die Monitor-Kommandos 10
4.2.2 Die Monitor-Routinen 13
4.3 Der Editor 17
4.3.1 Die Editor-Kommandos 19
4.4 Der Assembler 20
4.4.1 Steueranweisungen und Befehle 21
4.4.2 Bedingtes Übersetzen / conditional assembly 30
4.4.3 RAM-Belegung beim Arbeiten mit Editor und Assembler 30
4.5 Datei-Organisation 31
4.6 Das Terminal-Programm 31

Die Hardware
5.1 Funktionsbeschreibung und Adressdekodierung 34
5.2 Speicherchips und Jumper 35
5.3 Hinweise für Selbstbauer 36
5.4 Teile-Liste 38
5.5 ... läuft nicht, was nun? 39
5.6 Die Schnittstellen 40
5.6.1 Funktion der Schnittstellen 40
5.6.2 Interrupt-Steuerung 45
5.6.3 Register der Schnittstellen-Bausteine 48
5.6.4 Pinbelegung der Schnittstellen 50

Anhang: Monitorlisting 52

1 Die C'T KAT-Ce

Die M68000 CPU von Motorola gehört zu den leistungsfähigsten Mikroprozessoren, die derzeit erhältlich sind. Um an dem daraus resultierenden Fortschritt teilzuhaben, mußten Sie in der Vergangenheit ein komplettes System kaufen und dafür auf den Urlaub verzichten. Für die meisten Computerbesitzer bedeutet es außerdem, auf dem Schreibtisch einen weiteren Rechner mit Monitor, Tastatur, Laufwerken usw. unterbringen zu müssen. Eine Ablösung ist meistens nicht möglich, denn die fehlende Kompatibilität würde einen Verlust der vielen schönen Programme bedeuten (auch wenn sie längst nicht alle gebraucht werden: it's nice to have).
Mit der Entwicklung der C'T CAT-Ce wurden diese Probleme elegant umgangen. Durch die kompromißlose Auslegung einerseits und die Beschränkung auf das Erforderliche andererseits, ist es gelungen, einen Rechner herzustellen, der die volle Rechenleistung der M68000 CPU entfesselt. So wurde der Datenbus über die volle Breite von 16 Bit realisiert und die statischen RAMs benötigen keine Wartezyklen. Die verwendeten Schnittstellen-IC's wurden speziell für die 68000-Familie entwickelt und ermöglichen, die enorme CPU-Leistung gebührend an den Mann zu bringen.
Für Spezialisten ist noch eine Analog-Schnittstelle nachrüstbar. Nach Einlöten zweier Sockel und einiger Kleinteile kann die Platine mit zwei weiteren ICs bestückt werden. Sie erhalten dann zusätzlich einen AD-Eingang und einen DA-Ausgang.
Insgesamt hat die C'T KAT-Ce folgende Schnittstellen:

drei parallele Ports zu je 8 Bit, wovon 2 zu einem 16 Bit Port gekoppelt werden können. Die Datenrichtung ist bitweise wählbar.

zwei V24-Schnittstellen deren Baudrate per Software einstellbar ist. Es sind Übertragungsgeschwindigkeiten von 300 bis 38400 Baud möglich.

optional ein AD-Eingang und ein DA-Ausgang. Die Auflösung beträgt je 8 Bit und die Wandlerzeit liegt bei 14 Mikrosekunden.

eine Expansionsschnittstelle zum direkten Anschluß von weiteren Schnittstellen.

All dies wurde möglich, weil das gesamte vorhande Computersystem weiter benutzt wird. Mit Hilfe einer schnellen Rechnerkopplung und eines von C'T gelieferten Programms wird Ihr Computer zum intelligenten Terminal umfunktioniert. Wenn Sie die beiliegende Diskette booten, werden Sie einen völlig neuen Rechner erleben.
Damit die Programmierung gleich beginnen kann, haben wir ein dickes Softwarepaket mitgeliefert. Suchen sie es nicht, es steckt bereits in den ROMs auf der KAT-Ce Platine. Lange Diskettenzugriffe sind also nicht erforderlich. Nach dem Booten erwartet Sie der Monitor mit vielen leistungsfähigen Kommandos und nützlichen Routinen. Ein komfortabler Editor und ein schneller Assembler stecken ebenfalls in diesen beiden IC's.
Wenn Sie komplexe Steuerungsaufgaben vorliegen haben, können Sie mit unserem System zunächst die Software entwickeln und dann die C'T KAT-Ce als Prozessrechner einsetzen. Wenn's nötig ist, auch als völlig autarkes System (aber bitte mit Netzteil).
Sollte die C'T KAT-Ce Ihnen noch nicht schnell genug sein, eine kleine Reserve gibt es noch. Durch den Austausch von einigen Bauteilen sind noch 25% Geschwindigkeit drin.
Die Hard- und Software haben wir über längere Zeit sorgfältig getestet. Sollten Sie dennoch einen Fehler finden, bitten wir um eine formlose aber präzise Fehlermeldung. Wir werden den Fehler dann so schnell wie möglich beheben. Eine Haftung für Schäden, die

durch die Fehlfunktion unseres Produktes entstehen, können wir leider nicht übernehmen.
Für betriebsbereit gelieferte Computer leisten wir ein halbes Jahr lang Garantie. In dieser Zeit reparieren wir alle Defekte, die bei sachgemäßer Anwendung des Rechners an diesem entstehen. Die Garantie erlischt, wenn unzulässige Eingriffe, insbesondere mit dem Lötkolben, vorgenommen werden.

im Juli 1986 M.T. Himmeröder

2.1 Spannungsversorgung und Anschluß des KAT-CE

Die C'T KAT-Ce benötigt zum ordnungsgemäßen Betrieb folgende Spannungen:

+ 5 Volt	+/- 5%	0,7 Ampere
+12 Volt	+/-10%	50 Milliampere
-12 Volt	+/-10%	50 Milliampere

Um die C'T KATCe anzuschließen, legen Sie bitte die Platine mit der Bestückungsseite nach oben auf eine nichtleitende Unterlage (HÖR ZU) und drehen sie so, daß die große V24-Buchse rechts oben und die CPU M68000 links unten liegt. Die Spannungsanschlüsse finden Sie dann am oberen Rand der Platine über dem rechten RAM-Baustein 6264/43256.
Die 10 Lötstifte haben von links nach rechts folgende Bedeutung:

+5V Masse +5V Masse +5V Masse +5V Masse +12V -12V

Schließen Sie hiervon mindestens die vier rechten Stifte an, die anderen bei einem erhöhten Strombedarf (Expansionsschnittstelle).
Unter gewissen Bedingungen ist ein Betrieb mit nur +5 Volt zulässig. Ohne Einschränkungen funktioniert dies, wenn der AD-Wandler und die seriellen Schnittstellen gar nicht benötigt werden. Ist hingegen die Kabellänge zwischen der C'T KAT-Ce und der korrespondierenden Schnittstelle nicht größer als 5 Meter und sind in dieser Empfangsbausteine vom Typ 1489 oder 75189 eingesetzt, so können die seriellen Schnittstellen – aber nicht der AD-Wandler – wie folgt genutzt werden:
Tauschen Sie auf dem Bord der C'T KAT-Ce die beiden Sendebausteine mit der Bezeichnung 1488 oder 75188 gegen TTL-IC's vom Typ 74LS00 aus und verbinden Sie die Anschlüsse für +12 Volt und -12 Volt mit +5 Volt. Die Pegel entsprechen dann zwar nicht mehr der RS 232 Definition, aber es funktioniert bis 19200 Baud fehlerfrei. Höhere Übertragungsraten haben wir noch nicht ausprobiert.
Soll die Verbindung mit dem Terminalrechner über eine serielle Schnittstelle erfolgen, so geschieht dies über die Buchse „SERIELL 1". Die Pinbelegung dieser 25 poligen V24-Buchse entspricht der Norm (Siehe Pinbelegung der Schnittstellen 5.8.4).
Rechts von den Anschlüssen für die Spannungsversorgung liegen die beiden Stifte für die Reset-Taste. Schließen Sie hier einen Taster an, der im Normalzustand (also ungedrückt) offen ist.

2.2 Verbindung mit dem APPLE II

Falls Sie die C'T KAT-Ce von Ihrem APPLE aus mit UCSD-Pascal betreiben wollen, benötigen Sie das Programm KATCE.CODE. Sie finden es auf der mitgelieferten Diskette. Für die ersten Gehversuche hat es sich als praktisch erwiesen, eine bootfähige Diskette zu erstellen und mit dieser alle Funktionen in Ruhe zu testen. Für diesen Zweck erstellen Sie bitte eine Diskette mit folgendem Inhalt:

SYSTEM.PASCAL
SYSTEM.MISCINFO
SYSTEM.APPLE
SYSTEM.FILER

Übertragen Sie auf diese Diskette noch das Programm KATCE.CODE. Wenn Sie möchten, daß das Terminalprogramm nach dem Booten automatisch gestartet wird, ändern Sie den Namen bitte in SYSTEM.STARTUP. Sie werden dann nach dem Einschalten des Rechners vom Hauptmenü der C'T KAT-Ce begrüßt.
Der FILER ist natürlich nicht lebensnotwendig, aber recht praktisch bei der Überprüfung der Diskette.

Nun zur Hardware ...

Bevor Sie das Gehäuse des APPLE öffnen, bitte unbedingt den Rechner und die übrige Peripherie ausschalten.
Deponieren Sie das serielle Interface in Slot 2 und verbinden Sie dann das Kabel der Schnittstelle mit der C'T KAT-Ce. Falls Sie das Gehäuse Ihres Rechners nicht täglich öffnen: Die 16K Karte steckt, von der Tastatur aus gesehen, ganz links in Slot 0. Dann kommt in der Regel das Printer-Interface und dann Slot 2.
Der Anschluß des Netzteils sollte, wie im betreffenden Kapitel beschrieben, bereits erfolgt sein.
Und nun die C'T KAT-Ce einschalten, Diskette einlegen, Augen schließen und Rechner einschalten ....
Nach einigen Sekunden sollte das Hauptmenü auf dem Schirm stehen.
Durchgeschlagene Sicherungen, Stichflammen und Qualmwolken deuten auf einen Fehler. Daß Sie bei der Installation der Schnittstelle nicht viel falsch machen konnten, sollten Sie im Fehlerfall die Aufmerksamkeit auf die Spannungversorgung der C'T KAT-Ce lenken. Mehr dazu im Absatz 5.5 ... läuft nicht, was nun.
Für PRODOS-Besitzer haben wir eine besonders elegante Anschlußmöglichkeit entwickelt. Wenn Sie keine serielle Schnittstelle haben, kaufen oder löten Sie sich das in der Zeichnung angegebene Kabel und schließen es am Game I/O Connector des APPLE an. Wenn Sie alles richtig gemacht haben, erscheint, nach dem Booten von PRODOS und dem Start des Terminalprogramms, das Hauptmenü. Wählen Sie zunächst 'D' und geben Sie dann Ihre Schnittstelle für die C'T KAT-Ce an. Definieren Sie im nächsten Menü noch das Druckerinterface und lassen Sie dann die Angaben mit '1' auf die Bootdiskette speichern. Danach sind Sie wieder im Hauptmenü und können nun mit 'K' die C'T KAT-Ce starten. Künftig brauchen Sie sich um die Schnittstellen nicht mehr zu kümmern.
Über das Hauptmenü können Sie mit einem kleinen Trick auch den PRODOS-Filer aufrufen. Geben Sie 'F' und warten Sie bis der Filer von der Diskette geladen ist. Da der Filer

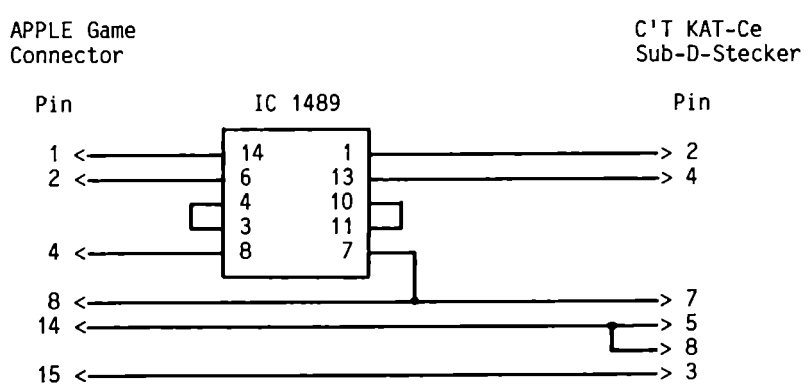
mit der 40 Zeichendarstellung arbeitet, müssen Sie dann mit Reset die 80 Zeichen-Karte verlassen. Keine Angst, der Filer verdaut das Reset ohne Probleme. Im Filer angelangt, werden Sie von Menü zu Menü weitergereicht. Alles selbstredend!
Das PRODOS-Terminalprogramm läßt auch die Definition eigener Kommandos und Treiberroutinen zu. Auf $202F steht die Endadresse des Programms, danach beginnt der freie Bereich. Auf $2003 steht ein JMP zum Einschleifen selbstgefertigter Kommandos. Der jeweilige Kommando-Code, siehe hierzu 4.6, ist im Akku vorhanden.
Ab $2007 beginnt die Adressleiste für die Treiberroutinen. Sie hat folgenden Inhalt (Low-Byte vorweg):

Adresse der Routine

$2007: Initialisierung der seriellen Schnittstelle
$2009: Empfangsroutine für die serielle Schnittstelle
$200B: Senderoutine für die serielle Schnittstelle
$200D: Initialisierung der Druckerschnittstelle
$200F: Senderoutine der Druckerschnittstelle
$2011: Initialisierung der Vidioausgabe
$2013: Videoausgabe
$2015: 80 Zeichenkarte abschalten

Wenn Sie hier die Adresse Ihrer eigenen Treiberroutine eintragen, wird künftig diese durchlaufen.
Zur Dateiorganisation unter PRODOS siehe 4.5

APPLE-PRODOS-Kabel



2.3 Verbindung mit COMMODORE C 64

Der C 64 wird an eine C'T KATCE über den Userport des C 64 und Port B der C'T KATCE angeschlossen. Achen Sie darauf, daß die Eproms der C'T KATCE für parallele Verbindungen vorgesehen sind.
Laden Sie folgendes File von der Diskette:
LOAD „KAT-CE",8,1
Anschließend wird das Terminalprogramm durch SYS 30000 gestartet.
Auf der Diskette werden die KATCE-Files mithilfe der Subdirectory KAT-CE/DIRECTORY verwaltet. Das Terminalprogramm legt dieses Subdirectory an, wenn es noch nicht auf der Diskette vorhanden ist. Die von der KATCE erzeugten Files können nur durch das Terminalprogramm bearbeitet oder auch angesehen werden. Filenamen dürfen bis zu 15 Zeichen lang sein, sie werden automatisch in Großbuchstaben überführt.
Verbindungskabel:

Userport unten		Port B KATCE
1	an	7
3	an	5
4	an	12
5	an	4
6	an	11
7	an	3
8	an	10
9	an	2
10	an	9

Bei dem Commodore C-64 fehlt eine ESC-Taste, drücken Sie stattdessen CTRL-Eckige Klammer auf. Außerdem erzeugt die LINKS-PFEIL-Taste einen falschen Code. Benutzen Sie CTRL-H, wenn Sie z.B. Eingabezeilen korrigieren wollen.

2.4 Verbindung mit dem GEPARD

Beim GEPARD ist die Sache ganz einfach, vorausgesetzt Sie haben eine Parallel-/Seriell-Interface-Karte vom Typ PS2. Das erforderliche Kabel und das Zwischenstück können Sie selbst anfertigen oder bei der Firma Himmeröder bestellen. Wenn Sie es selbst machen, benötigen Sie einen 26 poligen Pfostenfeldstecker mit Schneidklemmanschluß, ca. einen Meter 25 poliges Flachbandkabel und eine 25 polige Sub-D-Buchse (RS232/V24), ebenfalls mit Schneidklemmanschluß.
Klemmen Sie den Pfostenfeldstecker und die Sub-D-Buchse so an, wie in der Zeichnung gezeigt und verbinden Sie das Kabel mit der Schnittstelle 'Seriell 2', das ist lt. Bestückungsplan der Baugruppe der Anschluß J6. Achten Sie darauf, daß das Kabel nach der Montage zum Baugruppenstecker zeigt (so wie die Pfeile auf dem Bestükkungsdruck angeben).
Für das Zwischenstück benötigen Sie zwei 25 polige Sub-D-Stecker. Diese sind wie folgt miteinander zu verbinden:

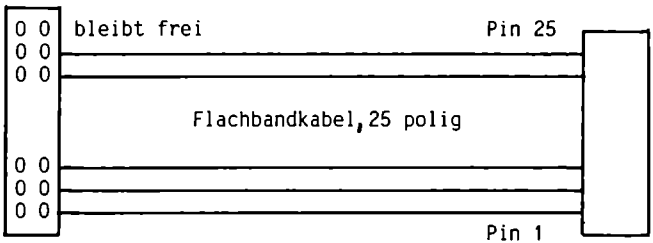
	Pin		Pin	
TxD	2	mit	3	RxD
RxD	3	mit	2	TxD
RTS	4	mit	5	CTS
CTS	5	mit	4	RTS
⊥	7	mit	7	⊥
DCD	8	mit	20	DTR
DTR	20	mit	8	DCD

Mit zwei 20 mm Distanzhülsen und zwei 30 mm Schrauben (M3) lassen sich die beiden Stecker zu einer kompakten Einheit zusammenschrauben.
Das Terminalprogramm finden Sie auf der Diskette.

Verbindungskabel GEPARD <--> C'T KAT-Ce

Pfostenfeldstecker
26 polig
von unten gesehen

Sub-D-Buchse
25 polig
von oben gesehen



2.5 Verbindung mit CPM-Rechnern

Bei der Vielfalt der CPM-Rechner können wir keine konkreten Anschlußpläne vorlegen. Für den PROF 80 haben wir eine laufende Version des Terminalprogramms vorliegen und stellen Ihnen den Source-Code auf Anforderung gerne zur Verfügung. Einen versierten Gesprächspartner können wir bei Bedarf vermitteln.

3 Aufteilung des Adressraumes

Direkt nach dem Einschalten und nach jedem Reset werden ab Adresse $0 die ROMs mit den Sprungvektoren, dem Monitor und dem Editor / Assembler eingeblendet. Sobald die CPU den Stackpointer geladen hat und mit der Ausführung des Programms

beginnen will, erfolgt dann eine Umschaltung der Speicherbereiche. Die ROMs werden nach $800000 verschoben und die RAMs sind ab $0 erreichbar. Das Initialisierunsprogramm kopiert dann zuerst die Vektoren ab $0 ins RAM. Sinn dieser recht komplizierten Anlaufprozedur ist es, die Vektoren mit einfachen MOVE-Befehlen ändern zu können.
Der RAM-Bereich beginnt also beim Ablauf eines Programms immer an der Adresse $0. Mit der empfohlenen Mindestausstattung von vier 6264-RAMs, endet der RAM bei $7FFF. Über mögliche andere RAM-Bestückungen gibt Abschnitt 5.2 Auskunft.
Serienmäßig beträgt die ROM-Größe der C'T KAT-Ce 16K Worte, aufgeteilt auf zwei 27128 ROMs von je 16K Bytes. Falls Ihnen diese ROMs nicht gefallen, vielleicht möchten Sie noch einige Routinen ergänzen, gibt Abschnitt 5.2 hierzu weitere Informationen.

Die Register des Port-Bausteins MC68230 beginnen bei $400001 und die Register des 'Doppel-UART' (DUART) MC68681 bei $402001.
Der Expansionsport ist unter $C00001 bis $C03FFF zu erreichen. Da er nur byteweisen Datenverkehr zuläßt, dürfen nur ungerade Adressen angesprochen werden. Näheres hierzu in Abschnitt 5.1.
Der AD-Wandler und der DA-Wandler belegen die Adressen $C04000, $C08000 und $C0C000.
Um die hardwaremäßige Aufteilung des Adressraumes einfach und schnell zu halten, wurden die Adressleitungen nur soweit wie erforderlich dekodiert. Dies hat zur Folge, daß die Speicher und Peripherie-Register auch unter anderen Adressen als den angegebenen zu erreichen sind. Genaueres hierzu unter 5.2.
Die Aufteilung des Adressraumes als Übersicht:

RAM-Speicher	2 X 6264: $000000 bis $003FFF
	4 X 6264: $000000 bis $007FFF
	2 X 43256: $000000 bis $00FFFF
	4 X 43256: $000000 bis $01FFFF
parallele Schnittstelle	$400001 bis $400035
serielle Schnittstelle	$402001 bis $40201F
EPROM-Speicher	2732: $800000 bis $801FFF
	2764: $800000 bis $803FFF
	27128: $800000 bis $807FFF
	27256: $800000 bis $80FFFF
	27512: $800000 bis $81FFFF
Expansionsschnittstelle	$C00001 bis $C03FFF
Digital-Analog-Wandler	$C04000
Analog-Digital-Wandler	$C08000 und $C0C000

4.1 Das Hauptmenü

Wenn die hardwaremäßige Verbindung der C'T KAT-Ce mit dem Terminalrechner korrekt funktioniert, erscheint das Hauptmenü auf dem Bildschirm unter folgenden Bedingungen:

1. nach dem Einschalten der Stromversorgung der C'T KAT-Ce (Power on Reset) und Start des Terminalprogramms im Terminalrechner.
2. nach Rücksetzen der C'T KAT-Ce (event. mehrfachem Drücken der Reset-Taste).
3. nach Beendigung eines Programms, welches vom Hauptmenü aus mit Xecute oder Run, oder nach einer Übersetzung vom Assembler aus gestartet wurde.
4. nach Beendigung des Editors oder des Assemblers, sofern keine andere Anweisung gegeben wurde.
5. wenn Sie den Monitor mit Q CR bedienen.

Unter der Kopfzeile des Hauptmenüs wird der RAM-Anfang und der Beginn des Assembler-Codes angezeigt. Beide Größen werden vom Monitor anhand der RAM-Größe errechnet.

Im Hauptmenü können Sie folgende Kommandos geben:

E Start des Editors.
Der RAM-Inhalt ist ohne Bedeutung und geht verloren.

W Start des Editors.
Ein im RAM vorhandener Text kann weiter bearbeitet werden.

M Aufruf des Monitors.

X Xecute/Ausführen eines Programms, das als Object-Code auf der Diskette gespeichert ist. Nach der Eingabe von X wird der Programmname erfragt. Er muß den Konventionen des Betriebssystems im Terminalrechner genügen. Programmlänge, Lade- und StartAdresse müssen am Beginn der Datei hinterlegt sein. Dies ist gewährleistet, wenn die Datei mit dem Monitorkommando SAVE erstellt wurde.

A Aufruf des Assemblers.
Der im RAM vorhandene Text wird Source-Code interpretiert und übersetzt. Der Text muß sich im Arbeitsbereich des Editors befinden.

R Ein im RAM vorhandenes Programm wird gestartet.
Der Editor-Assembler ermöglicht es dem Anwender, den Source- und Object-Code kleinerer Programme speicherresident zu halten und somit ständig zwischen Editor, Assembler und dem zu testenden Programm hin und her zu springen, ohne daß langwierige Diskettenzugriffe notwendig werden. Bedingung hierfür ist, daß das Programm nicht segmentiert ist und an der vom Monitor bestimmten Startadresse (CODESTART) beginnt. Der Quell-Code des Programms darf also weder eine ORG- noch eine CHAIN-Anweisung enthalten.

Q Die Kontrolle wird an das Betriebssystem des Terminalrechners zurückgegeben.

4.2 Der Monitor

Der Monitor bildet zusammen mit dem Programm im Terminalrechner das Betriebssystem der C'T KAT-Ce. Der Monitor enthält alle Routinen, die für den Datenverkehr mit dem Terminalrechner erforderlich sind, sowie die Routinen, die für ein effektives Entwickeln und Testen von Programmen gebraucht werden. Eine Beschreibung der realisierten Kommandos und Routinen finden Sie im weiter hinten. Wenn Sie einen Rechner kennen, dessen Monitorroutinen direkt angesprungen werden (z.B. APPLE), kennen Sie auch die daraus resultierenden Probleme. Fehlerbeseitigung und Verbesserung dieser

Routinen ist ohne Rückwirkung auf die Programme des Anwenders nicht möglich. Um diese Schwierigkeiten gar nicht erst aufkommen zu lassen, werden in der C'T KAT-Ce alle Routinen über eine Sprungliste adressiert. Diese ist am Beginn des Adressraumes, direkt hinter den hardwaremäßig ausgewerteten Vektoren eingerichtet unddamit sicher vor Verschiebungen.
In den ersten 256 Bytes, also im Adressraum $00 bis $FF, sind die Adressen hinterlegt, die der Prozessor beim Auftreten einer Ausnahmesituation auswertet. Mit Hilfe dieser Technik ist es möglich, auf fast alle Unregelmäßigkeiten per Programm zu reagieren. Wird z.B. in einem Programm durch Null dividiert, so erkennt der Prozessor den Fehler und führt die Routine aus, deren Adresse in $14 steht.
Wenn Sie nun nicht wollen, daß Ihr Programm bei einer Division durch Null beendet wird und Sie andererseits auch nicht jeden Divisor prüfen wollen, so können Sie den Adresspointer in $14 auf Ihre eigene Fehlerroutine „verbiegen".
Etwa so: MOVE.L #DIVNULL,$14 ; Adr.(DIVNULL) ⧫ $14
Eine Besonderheit stellt in diesem Zusammenhang der Befehl TRAP #n dar. Er erlaubt es, sozusagen per Software, einen Ausnahmezustand zu erzeugen.
Für TRAP #0 bis TRAP #15 sind die Adresszeiger in $80 bis $BF hinterlegt. Zwei dieser TRAPs sind bereits voreingestellt. Durchläuft ein Programm den Befehl TRAP #1, so wird die TRACE-Routine eingeschaltet. TRAP #0 schaltet sie wieder aus. Man erreicht so, daß nur die interessanten Befehle auf dem Schirm erscheinen und nicht die übrigen vielen Millionen. Natürlich können Sie auch die diesen TRAPs zugeordneten Sprünge umleiten, nur mit dem partiellen TRACE ist es dann vorbei. Mehr zum TRACE beim entsprechenden Monitorkommando.

4.2.1 Die Monitor-Kommandos

DISPLAY	ANF-Adresse . END-Adresse CR oder ANF-Adresse X Anzahl CR Der durch ANF-Adresse . END-Adresse oder ANF-Adresse X Anzahl definierte Bereich wird als Tabelle in HEX und ASCII angezeigt. Jeweils 8 Worte sind in einer Zeile wie folgt zusammengefaßt: Adresse: HEX-Wort HEX-Wort ... ASCII Ein Punkt bedeutet, daß das jeweilige Byte als ASCII-Zeichen nicht darstellbar ist. Die Ausgabe kann mit einer beliebigen Taste angehalten, dann mit CTRL C beendet oder mit jeder anderen Taste fortgesetzt werden. Die Adressen müssen gerade sein.
REGISTER DISPLAY	R CR Es werden die Inhalte der Register in HEX angezeigt, die mit dem Kommando OUTPUT CONTROL ausgewählt wurden. Erfolgte diese Auswahl nicht, werden die Register D0 bis D7 und A0 bis A7 angezeigt. Die letzte Zeile enthält den Inhalt des Status-Registers.
CHANGE	ANF-Adresse: HEX-Wort HEX-Wort .... CR Die angegebenen HEX-Werte werden ab ANF-Adresse wortweise im Speicher abgelegt. Als Trennzeichen zwischen zwei Worten ist ein

SPACE anzugeben. Sind mehr als 4 HEX-Zeichen für ein Wort angegeben, so gelangen nur die letzten 4 in den Speicher. Sind weniger als 4 HEX-Zeichen für ein Wort angegeben, so werden diese im Wort rechtsbündig abgelegt. Die führenden Nibble enthalten in diesem Fall Nullen. Die Adresse muß gerade sein.

MOVE — ANF-Adresse . END-Adresse M ZIEL-Adresse CR

Der Bereich von ANF-Adresse bis END-Adresse einschl. wird wortweise in den Bereich ab ZIEL-Adresse übertragen. Alle 3 Adressen müssen gerade sein.

VERIFY — ANF-Adresse . END-Adresse V VRY-Adresse CR

Der Bereich von ANF-Adresse bis END-Adresse einschl. wird wortweise mit dem Bereich ab VRY-Adresse verglichen. Unterschiede werden als Tabelle von

Adresse:Inhalt Adresse:Inhalt

angezeigt. CTRL C beendet die Ausgabe vorzeitig. Alle 3 Adressen müssen gerade sein.

DISASSEMBLE — ANF-Adresse Z CR oder

ANF-Adresse Z+ CR

Ab ANF-Adresse wird der Speicherinhalt als Programm interpretiert und angezeigt. Jeder Befehl wird in einer Zeile der Form

Adresse: Befehl Operand Object-Code ASCII

ausgegeben. Für nicht interpretierbare Befehle erscheint *****, für nicht darstellbare ASCII-Zeichen ein Punkt. Z ohne nachfolgendes + bewirkt die Ausgabe von 16 Befehlen, Z+ die fortlaufende Ausgabe. Im Falle von Z+ kann die Ausgabe mit einer beliebigen Taste angehalten, dann mit CTRL C beendet oder mit jeder anderen Taste fortgesetzt werden. Die Adresse muß gerade sein.

START — ANF-Adresse G CR

Das Programm im Speicher wird bei ANF-Adresse gestartet. Die Adresse muß gerade sein.

TRACE — ANF-Adresse T CR

Das Programm im Speicher wird bei ANF-Adresse gestartet. Für jeden durchlaufenen Befehl wird eine Zeile der Form

Adresse: Befehl Operand Object-Code ASCII

ausgegeben. Ein Punkt bedeutet, daß das jeweilige Byte als ASCII-Zeichen nicht darstellbar ist. Zusätzlich werden die Inhalte der Register in HEX angezeigt, die mit dem Kommando OUTPUT CONTROL ausgewählt wurden. Erfolgte diese Auswahl nicht, werden die Register D0 bis D7 und A0 bis A7 angezeigt. Die jeweils letzte Zeile enthält den Inhalt des Status-Registers. Nach dem Start befindet sich das Programm im Einzel-Step-Modus. D.h. nach jedem Befehl wird auf den Druck einer Taste gewartet. CTRL C beendet das Programm, CR löscht den Einzel-Step-Modus und jede andere Taste setzt das Programm im Einzel-Step-Modus fort. Die Adresse muß gerade sein.

OUTPUT CONTROL — O D0 .. Dn .. D7 A0 .. An .. A7 CR

Die mit D0 bis D7 und/oder A0 bis A7 ausgewählten Register werden bei TRACE angezeigt. Die Ausgabe der Register erfolgt, unabhängig von der Auswahl, immer in aufsteigender Folge (Dn < An).

SAVE

S Name, ANF-Adresse, Länge CR

Die Anweisung speichert den Bereich (ANF-Adresse,Länge) in den File 'Name'. Die explizite Angabe des Laufwerkes kann gemäß den Konventionen des jeweiligen Betriebssystems im Terminal-Rechner erfolgen. ANF-Adresse und Länge werden in zwei L-Worten am Beginn des Files gespeichert. Die Adresse muß gerade sein.

LOAD

L Name,ANF-Adresse CR oder

L Name,ANF-Adresse G CR

Der Inhalt des Files 'Name' wird ab ANF-Adresse in den Speicher geladen. Wenn die Angabe der Ladeadresse unterbleibt, wird sie, wie die Länge, den ersten beiden L-Worten des Files 'Name' entnommen. Nach dem Laden wird die Ladeadresse mit der Meldung

Anfang bei: Adresse

angezeigt. Die explizite Angabe des Laufwerkes kann gemäß den Konventionen des jeweiligen Betriebssystems im Terminalrechner erfolgen. Die zusätzliche Angabe von ',G' bewirkt, daß das Programm nach dem Laden gestartet wird. In diesem Fall unterbleibt die Anzeige der Startadresse.

AUSGABE

P1 CR, P2 CR, P3 CR, P4 CR oder P0 CR

P1 CR initialisiert Port A. Alle Daten die die C'T KAT-Ce zum Terminalrechner sendet und die dort in der Regel auf dem Bildschirm erscheinen, werden zusätzlich an Port A ausgegeben. Die Initialisierung erfolgt so, daß die Anforderungen einer CENTRONICS-Druckerschnittstelle erfüllt werden.

P2 CR veranlaßt, daß alle zum Terminalrechner gesendeten Daten auch auf dem Drucker des Terminalrechners ausgegeben werden.

P3 CR initialisiert die Schnittstelle SERIELL-A. Es handelt sich dabei um die aus 10 Pfostenfeldstiften bestehende Schnittstelle unterhalb der SUB-D-Buchse. Alle Daten, die die C'T KAT-Ce zum Terminalrechner sendet, werden zusätzlich an dieser Schnittstelleausgegeben. Bevor dieses Kommando gegeben wird, muß in die Speicherstelle $301 der zugehörige Baudratenwert eingetragen werden (s.4.2.2 SERAINIT).

P4 CR initialisiert die Schnittstelle SERIELL-B. Es handelt sich dabei um die 25polige SUB-D Buchse. Alle Daten, die die C'T KAT-Ce zum Terminalrechner sendet, werden zusätzlich an dieser Schnittstelleausgegeben. Bevor dieses Kommando gegeben wird, muß in die Speicherstelle $300 der zugehörige Baudratenwert eingetragen werden (s.4.2.2 SERAINIT). Falls Sie den Hostrechner über diese Schnittstelle angeschlossen haben, dürfen Sie das Kommando P4 natürlich nicht geben.

P0 CR beendet die Druckausgabe.

RECHNEN

? A Operator B

Die Operanden A und B werden gemäß Operator wortweise miteinander verknüpft. Zulässig sind + - * und / für die vier Grundrechnungsarten. Die Operanden A und B werden als HEX-Zahlen ohne Vorzeichen interpretiert. Bei der Division wird das Ergebnis in der Form Quotient & Rest dargestellt.

QUIT Q CR
Es wird zum Hauptmenü verzweigt.

4.2.2 Die Monitor-Routinen

KEYPRESS
$800110 veranlaßt eine Abfrage der Tastatur im Terminalrechner. Wurde ein Zeichen eingegeben, gelangt Eins, anderenfalls Null in das niederwertige Byte von D0. Bit 1 bis 31 werden Null gesetzt. Außer D0 werden keine Register verändert.

BYTEIN
$800114 liest ein Byte von der Schnittstelle zum Terminalrechner und übergibt es im niederwertigen Byte von D0. Außer D0 werden keine Register verändert.

TASTIN
$800118 liest ein Zeichen von der Tastatur und übergibt es im niederwertigen Byte von D0. DasMSB und die Bits 31 bis 8 sind immer Null. Außer D0 werden keine Register verändert.

GETSTR
$80011C liest einen String ein und legt ihn an die durch Register A1 bezeichnete Stelle im RAM ab. Das erste Byte enthält die Anzahl der eingelesenen Zeichen. Das abschließende CR wird übernommen, ist aber in der Anzahl nicht enthalten. Es werden keine Register verändert.

BYTEOUT
$800120 sendet das niederwertige Byte in D0 über die Schnittstelle zum Terminalrechner. Es werden keine Register verändert.

WORDOUT
$800128 gibt die beiden Bytes im niederwertigen Wort von D0 über das Unterprogramm BYTEOUT aus. Register D0 wird verändert.

ASCIOUT
$800124 stellt die beiden Zeiger XKOORD und YKOORD und gibt anschließend das niederwertige Byte in D0 über das Unterprogramm BYTEOUT auf dem Bildschirm aus. Die beiden Zeiger geben an, in welcher Spalte (XKOORD) und Zeile (YKOORD) das nächste Zeichen auf dem Bildschirm angezeigt wird. XKOORD ist unter der Adresse $3AD und YKOORD unter der Adresse $3AC zu erreichen; beide Werte sind je ein Byte lang. Der Zeiger XKOORD wird nur bei der Ausgabe von CR auf Null zurückgestellt. Über die eingestellte Zeilenlänge hinaus ausgegebene Zeichen werden ignoriert und gehen verloren. YKOORD bewegt sich immer im Bereich 0 bis 23. Die Ausgabe von CTRL L löscht den Bildschirm und stellt XKOORD und YKOORD auf Null. Es werden keine Register verändert.

STROUT
$800134 gibt eine Zeichenfolge (String) über das Unterprogramm ASCIOUT aus. Das erste Byte des Strings muß die Anzahl der auszugebenden Zeichen

enthalten. Die Adresse des 'Strings' ist in A1 vorzugeben. Die Register D0, D1 und A1 werden verändert. Hinweis: Wird eine Zeichenfolge im Assembler-Source-Code mit der Anweisung STR definiert, so setzt der Assembler dem Text die Stringlänge automatisch voran.

PRTBYTE
$80012C konvertiert die beiden Nibble im niederwertigen Byte von D1 in HEX-Zeichen und gibt sie über das Unterprogramm ASCIOUT aus. Das Register D0 wird verändert.

PRTWORD
$800130 gibt die beiden Byte im niederwertigen Wort von D1 über das Unterprogramm PRTBYTE aus. Register D0 wird verändert.

CROUT
$800138 gibt über das Unterprogramm ASCIOUT ein CR und ein LF aus. Das Register D0 wird verändert.

CLBILD
$80013C gibt über das Unterprogramm ASCIOUT ein CTRL L aus und bewirkt damit das Löschen des Bildschirmes. Die Zeiger XKOORD und YKOORD werden auf Null gestellt. Das Register D0 wird verändert.

PRTCOL
$800140 gibt über das Unterprogramm ASCIOUT einen Doppelpunkt aus. Das Register D0 wird verändert.

PRTMSP
$800144 gibt über das Unterprogramm ASCIOUT eine Folge von Blank (Hex 20) aus. Im Register D1 ist die Anzahl-1 vorzugeben. Die Register D0 und D1 werden verändert.

GOTOXY
$800148 positioniert den Cursor auf dem Bildschirm in Spalte X der Zeile Y. Die Vorgabe der Koordinaten erfolgt im niederwertigen Wort von D1. Im ersten Byte ist der Wert für Y und im zweiten Byte der Wert für X anzugeben. X,Y = 0 stellt den Cursor in die linke obere Ecke. Die Zeiger XKOORD und YKOORD (siehe ASCIOUT) werden auf X,Y gestellt. Es werden die Register D0 und D2 verändert.

BELL
$80014C gibt über das Unterprogramm ASCIOUT CTRL-G aus. Ein APPLE-Terminal z.B. macht dann 'tröööt'. Das Register D0 wird verändert.

QUIT
$80010C JMP QUIT beendet das Programm und gibt die Kontrolle an den Terminal-Rechner zurück. Falls Sie in der KAT-Ce bleiben wollen, muß das Programm mit RTS beendet werden.

REWRITE
$800158 eröffnet eine neue Datei auf der Diskette. Laufwerk und Dateiname werden gemäß den Konventionen des Betriebssystems im Terminalrechner als String (siehe STROUT) definiert. Die Stringadresse ist in Register A1 vorzugeben. Damit bei den übrigen Diskettenbefehlen nicht immer der ganze String untersucht werden muß, wird diesem bei der Dateieröffnung für die Dauer der Bearbeitung eine File-nummer zugeordnet. Die Filenummer ist in Register D1 vorzugeben. Die höchste Filenummer ist bestimmt durch die Anzahl der Files, die das Betriebssystem im Termi-

nalrechner gleichzeitig in Bearbeitung halten kann. Die niedrigste Filenummer ist Null. Ist auf dem ausgewählten Laufwerk bereits eine Datei gleichen Namens vorhanden, wird diese zuvor gelöscht. Es werden die Register D0, D2 und A1 verändert.

RESETF
$800154 eröffnet eine bereits vorhandene Datei auf der Diskette. Die Parameter in den Registern A1 und D1 haben die gleiche Bedeutung wie beim Aufruf von REWRITE. Es werden die Register D0, D2 und A0 verändert.

BLOCKR
$80015C liest von der Diskette aus der Datei mit der Filenummer lt. Register D1 den Block mit der Blocknummer lt. Register D5 (wahlfreier Zugriff). Die Daten werden im Diskbuffer ab Adresse $100 abgelegt. Die Blocklänge beträgt konstant 512 Bytes. Der erste Block einer Datei hat die Blocknummer Null. Es werden die Register D0, D2 und A0 verändert.

BLOCKW
$800160 schreibt einen Block von 512 Bytes in die Datei mit der Filenummer lt. Register D1. Die Blocknummer in Register D5 gibt an, welchen Platz der Block in der Datei erhält (wahlfreier Zugriff). Null bezeichnet den ersten Block einer Datei. Die Daten müssen im Diskbuffer ab Adresse $100 stehen. Es werden die Register D0, D2 und A0 verändert.

CLOSE
$800164 schließt die Datei mit der Filenummer lt. Register D1 und gibt die im Terminalrechner für diese Datei belegten Bereiche frei. Es ist notwendig, eröffnete Dateien nach Abschluß der Bearbeitung wieder zu schließen. Es werden die Register D0 und D2 verändert.

CLOSEDEL
$800168 schließt und löscht die Datei mit der Filenummer lt. Register D1 und gibt die im Terminalrechner für diese Datei belegten Bereiche frei. Es werden die Register D0 und D2 verändert.

IORESULT
$80016C liefert in Register D0 den Return-Code der Disketten-Befehle. Die Fehlernummer ist abhängig vom Betriebssystem im Terminalrechner. Das Register D0 ist Null, wenn der Befehl fehlerfrei ausgeführt wurde.

UNITOUT
$800170 sendet das niederwertige Byte in D0 über die Schnittstelle zum Drucker des Terminalrechners. Es werden keine Register verändert.

UNITINIT
$800174 veranlaßt die Initialisierung des Druckers vom Terminalrechner. Es werden keine Register verändert.

DAOUT
$800178 gibt das niederwertige Byte in D0 über den Digital-Analog-Wandler aus. Es werden keine Register verändert.

ADIN
$80017C startet den Analog-Digital-Wandler und übergibt den Messwert im niederwertigen Byte von D0. Außer D0 werden keine Register verändert.

PARAINIT
$800180 initialisiert die parallele Schnittstelle Port A. Alle Datenbits werden als Ausgang definiert. Die Leitung H1 wird so geschaltet, daß sie einen

ankommenden Low-High Übergang als Handshake-Signal interpretiert. Die Leitung H2 dient zur Erzeugung eines negativen Strobe-Impulses. Durch die Ausgabe von CR wird die Gegenseite aufgefordert zur Bestätigung ein Handshake-Signal auszugeben und damit ihre Empfangsbereitschaft zu signalisieren. Port A entspricht damit den Anforderungen einer CENTRONICS-Schnittstelle. Das Register D0 wird verändert.

PARAOUT
$800188 Wenn der Empfang des zuletzt gesendeten Bytes von der Gegenseite bestätigt wurde, wird das niederwertige Byte von Register D0 über Port A ausgegeben. Anderenfalls wird zuvor auf die Bestätigung gewartet.

SERAINIT
$8001A0 Die serielle Schnittstelle SERIELL1 wird für das Übertragungsformat STARTBIT, 8 DATENBIT und 1 STOPBIT ohne Parity initialisiert. Die Übertragungsgeschwindigkeit (Baudrate) wird im niederwertigen Byte vonRegisterD1 für den Sender und Empfänger getrennt eingestellt. Bit 7 bis 4 sind dem Empfänger und Bit 3 bis 0 dem Sender zugeordnet. Die nachfolgende Tabelle zeigt die Codierung der Baudraten.

Baudrate	Code
75	$0
110	$1
134.5	$2
150	$3
300	$4
600	$5
1200	$6
2000	$7
2400	$8
4800	$9
1800	$A
9600	$B
19200	$C
38400	$D

Enthält Register D1 beim Aufruf von SERAINIT z.B. $60, so werden die Daten von der C'T KAT-Ce mit 75 Baud gesendet und mit 1200 Baud empfangen. Außer D1 werden keine Register verändert.

SERAOUT
$8001A8 Sobald die Schnittstelle SERIELL1 frei ist, wird das niederwertige Byte von Register D0 an SERIELL1 ausgegeben. Außer D0 werden keine Register verändert.

SERAIN
$8001AC Von der Schnittstelle SERIELL1 wird ein Byte nach Bit 7 bis 0 in Register D0 gelesen. Falls noch keine Daten empfangen wurden, wird auf deren Eintreffen gewartet. Außer D0 werden keine Register verändert.

SERBINIT
$8001B0 Die serielle Schnittstelle SERIELL2 wird initialisiert. Im übigen gelten die bei SERAINIT gemachten Angaben.

SERBOUT
$8001B8 Sobald die Schnittstelle SERIELL2 frei ist, wird das niederwertige Byte

von Register D0 an SERIELL2 ausgegeben. Außer D0 werden keine Register verändert.

SERBIN

$8001BC Von der Schnittstelle SERIELL2 wird ein Byte nach Bit7 bis 0 in Register D0 gelesen. Falls noch keine Daten empfangen wurden, wird auf deren Eintreffen gewartet. Außer D0 werden keine Register verändert.

INITPORTB

$8001C0 Die parallele Schnittstelle Port B wird zur Datenübertragung in zwei Richtungen initialisiert:
Für jede Richtung werden je zwei Leitungen als Datenleitung und zwei Leitungen für Handshakesignale definiert.

READPORTB

$8001C4 liest ein Byte von Port B in das niederwertige Byte von Register D0. Falls noch keine Daten anstehen, wird auf deren Eintreffen gewartet. Außer D0 werden keine Register verändert.

SENDPORTB

$8001C8 Wenn der Empfang des zuletzt gesendeten Bytes von der Gegenseite bestätigt wurde, wird das niederwertige Byte von Register D0 über Port B ausgegeben. Anderenfalls wird zuvor auf die Bestätigung gewartet. Außer D0 werden keine Register verändert.

4.3 Der Editor

Für die Erstellung von Programmtexten ist in den ROMs ein Fullscreen-Editor enthalten. Dieser ist in der Bedienung weitgehend identisch mit dem Editor von TURBO-PASCAL und das, obwohl wir ihn ganz neu geschrieben haben. Wenn Sie also WORD-STAR oder TURBO-PASCAL bereits kennen, brauchen Sie nicht umzulernen. Kennen Sie nur einen Zeileneditor, werden Sie bald die Vorzüge eines Fullscreen-Editors zu schätzen wissen.
Um zügiges Editieren zu gewährleisten, wurde die Textgröße auf den zur Verfügung stehenden RAM-Bereich beschränkt. Denken Sie also rechtzeitig daran, Ihr Programm in RAM-gerechte Module zu unterteilen. Mit dem ebenfalls im ROM untergebrachten Assembler können Sie dann die einzelnen Module zu einem Programm zusammenfassen.
Wenn Sie vom Hauptmenü aus mit E den Editor aufrufen, ist zuerst die Frage 'Datenladen? (J/N)' zu beantworten.

Wenn Sie mit J antworten, müssen Sie anschließend den Dateinamen des einzulesenden Textes angeben. Dieser Name muß Laufwerk und Dateinamen für das Programm im Terminalrechner eindeutig spezifizieren. In der C'T KAT-Ce erfolgt keine Prüfung des Namens. Ist die Datei auf der Diskette vorhanden, wird der Text eingelesen und der Bildschirm zeigt die ersten 23 Zeilen des Textes an.

Wenn Sie mit N antworten, wird unterstellt, daß ein neuer Text angelegt werden soll. Der hierfür vorgesehene RAM-Bereich wird gelöscht und der Editor ist für die Eingabe bereit.

Wenn Sie den Editor vom Hauptmenü aus mit W aufrufen, so wird angenommen, daß der zu bearbeitende Text bereits im RAM vorhanden ist und Sie können deshalb sofort

mit Editieren anfangen bzw. weitermachen. Sollte noch kein Text im RAM vorhanden sein, reagiert das Programm undefiniert. Ein Systemabsturz kann die Folge sein.
Die oberste Zeile auf dem Bildschirm ist die Statuszeile. Hier wird angezeigt, ob sich der Editor im Einfüge- oder im Überschreibmodus befindet. Wenn der Einfügemodus eingeschaltet ist, werden alle eingegebenen Zeichen an der vom Cursor bezeichneten Stelle eingefügt und der nachfolgende Text wird jeweils um ein Zeichen aufgeschoben. Im Überschreibmodus hingegenwerden die Zeichen unter dem Cursor überschrieben. Zwischen Einfüge- und Überschreibmodus können Sie mit Control V hin und her schalten. Beim ersten Aufruf des Editors und nach dem Einlesen eines Textes von der Diskette ist immer der Einfügemodus eingestellt.
Mit Control QV kann der Einrückmodus an- und ausgeschaltet werden. Ist der Einrückmodus an, beginnt eine neue Zeile immer in der gleichen Spalte wie die vorhergehende Zeile. Ist der Einrückmodus aus, beginnt eine neue Zeile immer in Spalte 1.
Im vorliegenden Editor sind 48 Kommandos realisiert. Diese lassen sich in folgende 4 Gruppen unterteilen:

Kommandos für die Cursor-Bewegung
Kommandos für Einfügen und Löschen
Kommandos zum Bearbeiten ganzer Textblöcke
Sonstige Kommandos

Da für die Bearbeitung eines Textes neben Buchstaben, Ziffern und Sonderzeichen noch viele Steuerzeichen erforderlich sind, der Editor aber auch mit bescheiden ausgestatteten Tastaturen laufen soll, wurde auf die Benutzung von Cursor-Tasten und dergl. verzichtet. Stattdessen wird für die Eingabe von Steuerzeichen mit der Control-Taste auf eine zusätzliche Ebene umgeschaltet. Da dann die Beschriftung der Tasten nicht mehr mit der Funktion übereinstimmt, macht man sich als Gedächtnisstütze die geometrische Anordnung der Tasten zunutze. Für die einfache Cursorbewegung sind dies die Tasten E S D X. Diese Tasten liegen so, daß sich die aus der Betätigung ergebende Cursorbewegung von selbst erklärt. Für größere Sprünge, nämlich um ein Wort nach links oder rechts, sind die jeweils daneben liegenden Tasten A und F vorgesehen. Über mehrere Zeilen auf und ab geht es dann mit R und C. Noch größere Sprünge werden durch die vorherige Eingabe von Control Q ausgelöst. S und D bewegen dann den Cursor zum Zeilenanfang bzw. Zeilenende und R und C zum Dateianfang bzw. Dateiende.
Control J und I veranlassen einen Tabulatorsprung nach links bzw. rechts. Tabulatoren sind alle 10 Spalten gesetzt. Sie stehen dort unveränderbar und sollen Ihnen helfen in Ihren Texten Ordnung zu halten.
Natürlich kann man nicht nur den Cursor über den Bildschirm flitzen lassen. Es läßt sich ebenso das durch den Bildschirm dargestellte Textfenster auf und ab bewegen. In Neudeutsch nennt man dies 'Scrollen'. Control W bewegt das Textfenster um eine Zeile nach oben, Control Z um eine Zeile nach unten. Control Z paßt übrigens nicht in dieses Schema, wenn Sie eine deutsche Tastatur besitzen. In Amerika hat man auf der Tastatur Z und Y vertauscht und dß der Ur-Editor dort entstanden ist, müssen wir aus Kompatibilitätsgründen mit diesem Schönheitsfehler leben. Im Editor wäre dies sehr leicht zu ändern, aber die vielen Handbücher zu ändern ist nicht vertretbar. Und erst die vielen Kopien ...
Da Sie als C'T KAT-Ce-Besitzer zur Elite der Computeranwender gehören, gehen wir davon aus, daß Sie die anderen Editorkommandos, anhand der nachfolgenden Aufstellung, auch ohne langatmige Beschreibung begreifen. Trockenschwimmen ist da ohnehin sinnlos. Am besten, Sie beginnen mit einem kleinen Programm.

4.3.1 Die Editor-Kommandos

Cursor Bewegungen:

Ctrl-S	Zeichen links (ohne Löschen)
Ctrl-H	Zeichen links (mit Löschen)
Ctrl-D	Zeichen recht
Ctrl-A	Wort links
Ctrl-F	Wort rechts
Ctrl-E	Zeile nach oben
Ctrl-X	Zeile nach unten
Ctrl-W	Scroll up
Ctrl-Z	Scroll down
Ctrl-R	Seite zurückblättern
Ctrl-C	Seite vorblättern
Ctrl-Q-S	zum Zeilenanfang
Ctrl-Q-D	zum Zeilenende
Ctrl-Q-E	zum Seitenanfang
Ctrl-Q-M	Seite zentrieren
Ctrl-Q-X	zum Seitenende
Ctrl-Q-R	zum Textbeginn
Ctrl-Q‹n›R	um n-Seiten zurück
Ctrl-Q-C	zum Textende
Ctrl-Q‹n›C	um n-Seiten vorwärts
Ctrl-Q-B	zum Blockbeginn
Ctrl-Q-K	zum Blockende
Ctrl-Q-P	zur letzten Cursorposition

Einfügen und Löschen:

Ctrl-V	Einfügemodus an/aus
Ctrl-N	Zeile einfügen
Ctrl-Y	Zeile löschen
Ctrl-Q-Y	Rest der Zeile löschen
Ctrl-G	Zeichen unter dem Cursor löschen
Ctrl-T	Wort rechts löschen

Blockkommandos:

Ctrl-K-B	Blockanfang markieren
Ctrl-K-K	Blockende markieren
Ctrl-K-T	Wort über dem Cursor markieren
Ctrl-K-C	Block kopieren
Ctrl-K-V	Block versetzen
Ctrl-K-Y	Block löschen
Ctrl-K-R	Block von Diskette lesen und vor der aktuellen Zeile einfügen
Ctrl-K-W	Block auf Diskette schreiben (Die beiden markierten Zeilen werden ganz mit abgespeichert)

Verschiedene Kommandos:

ESCAPE	Editieren Ende
Ctrl-Q-V	Einrückmodus an/aus
Ctrl-I	Tab rechts
Ctrl-J	Tab links
Ctrl-Q(n)I	Tab-Distanz setzen
Ctrl-Q(n)F	String bzw. Wort suchen
Ctrl-Q(n)A	String bzw. Wort suchen und ersetzen
Ctrl-P	Ersatztaste für Control-Zeichen definieren
Ctrl-L	Wort über Cursor suchen
Ctrl-Q-L	Letzte Suche wiederholen
Ctrl-Q-T	Zeilenlänge auf Bildschirmbreite reduzieren

4.4 Der Assembler

Den ROM-residenten 2-Pass-Assembler erreichen Sie vom Hauptmenü aus mit A oder beim Verlassen des Editors nach 'bejahen' der entsprechenden Frage. 2-Pass heißt hier, daß die Umwandlung eines Quelltextes in ein Maschinenprogramm in 2 Durchläufen erfolgt. Dabei muß jeweils der ganze Text von Anfang bis Ende analysiert werden. Der Assembler beginnt den ersten Pass einer Übersetzung generell mit dem im Textbereich des Editors stehenden Quelltext. Findet der Assembler in diesem Text keine Anweisung zum Nachladen eines weiteren Textes, so laufen erster und zweiter Pass ohne Diskettenzugriffe direkt hintereinander ab. Die Übersetzungszeit beträgt dann nur einige Sekunden.

Den Quelltext eines Programms kann man aus organisatorischen Gründen, oder weil die RAM-Größe dies erfordert, auf mehrere Diskettendateien verteilen. Eine Nachladeanweisung (CHAIN) am Ende eines 'Teil-Textes' teilt dem Assembler dann mit, welche Datei als nächstes einzulesen ist. Um Platz zu sparen, wird hierfür der Textbereich des Editors mehrfach genutzt, oder besser gesagt: überschrieben. Es muß also, wenn der erste Pass zu Ende ist, die erste Textportion – die ursprünglich im RAM stand – von der Diskette gelesen werden. Dies erfordert, daß der erste Textteil – der für den Start der Übersetzung nur im RAM stehen mußte – auch auf der Diskette zu finden ist. Den Dateinamen dieser ersten Datei sucht der Assembler im Editor. Es ist also zwingend erforderlich, diese erste Datei vor dem Beginn einer Übersetzung mit dem Editor auf die Diskette zu schreiben oder von ihr zu lesen.

Die vom Assembler erzeugten Maschinenbefehle (Object-Code) werden in Blöcken von 512 Bytes auf der Diskette abgespeichert. Die Programmgröße ist bei diesem Verfahren also nur durch die RAM-Größe und die Anzahl der Label beschränkt. Näheres hierzu finden Sie unter 4.4.3. Die Startadresse des Programms und der Dateinamefür den Object-Code sind in einer ORG-Anweisung anzugeben. Der Assembler speichert die Startadresse und die Programmlänge in je zwei L-Worten am Beginn der Datei.

Findet der Assembler im Quellcode einen Fehler, so wird eine Fehlermeldung ausgegeben und auf einen Tastendruck gewartet. Durch die Eingabe von E wird dann die Übersetzung abgebrochen, der Editor aufgerufen, der Quelltext ausgegeben und der Cursor

auf das Ende der fehlerhaften Zeile positioniert. Sie brauchen dann nur noch den Fehler zu korrigieren und erneut den Assembler zu starten. Besteht das Programm aus mehreren Dateien, müssen Sie zuvor natürlich die Start-Datei laden, so wie im vorherigen Abschnitt beschrieben.
Der Quelltext braucht für den Assembler nicht formatiert zu sein. Ein Blank zwischen Label und Befehl bzw. Befehl und Operand reicht aus. Eine Zeile ohne Label muß mit einem Blank beginnen. Die Label-Länge ist im Prinzip nicht begrenzt, der Assembler wertet aber nur die ersten 9 Stellen für die Unterscheidung der Label aus. Ein Label muß mit einem Buchstaben beginnen.
Nun noch etwas zur Syntax. Operanden werden entweder als Register oder als Ausdruck angegeben. Als Register sind die Datenregister D0 bis D7 und die Adressregister A0 bis A7 sowie SR und CCR zulässig. SR bezeichnet das Statusregister und CCR das Condition-Code-Register. A7 gibt es doppelt. Abhängig vom S-Bit (Supervisor-Status) im SR wird A7 oder A7' angesprochen. Da A7 als Stackpointer dient, gibt es somit einen Supervisorstackpointer und einen User-Stackpointer. In der C't KAT-Ce ist das S-Bit immer gesetzt. Wenn Sie es nicht ändern, können Sie es vergessen.
Ein Ausdruck besteht aus einem oder mehreren Elementen, die durch + oder - verknüpft sein können. Als Elemente sind zulässig:

symbolische Namen
Dezimalzahlen
Hexadezimalzahlen (beginnend mit "$")
Binärmuster (beginnend mit "%")
Texte (zwischen Apostrophen)

Beispiele für syntaktisch richtige Ausdrücke sind:

```
NAME1-NAME2
NAME1+100
NAME2+$10F
$100-100+'A'
NAME1+%10101011
```

Ein Ausdruck wird vom Assembler während der Übersetzung errechnet und im Object-Code abgelegt. Ziel der ganzen Operation ist es, eine sinnvolle Adresse zu bestimmen. Der Ausdruck NAME 1+NAME 2 ist zwar auch errechenbar, stellt aber keine sinnvolle Adresse dar, wenn NAME 1 und NAME 2 Speicheradressen sind.

4.4.1 Steueranweisungen und Befehle

Außer den Befehlen, die der Assembler direkt in Maschinenbefehle umsetzt, gibt es noch eine Reihe von Steueranweisungen die den Assembler mit zusätzlicher Information versorgen, die Definition von Konstanten ermöglichen, den Übersetzungslauf beeinflussen oder die Protokollausgabe steuern. Diese Steueranweisungen werden Pseudo-Befehle genannt, weil sie im Quelltext an der gleichen Stelle stehen müssen wie die Befehle aber selbst keinen Object-Code ergeben. Folgende Pseudo-Befehle sind im Assembler realisiert:

EQU — Diese Anweisung ordnet einem symbolischen Namen (Label) einen Ausdruck zu. Weil das Ergebnis einer EQU-Anweisung die Länge nachfol-

gender Befehle beeinflussen kann, muß es im ersten Pass vollständig ermittelt werden. Der Ausdruck einer EQU-Anweisung darf deshalb neben Konstanten nur symbolische Namen enthalten, die bereits vorher definiert wurden.
EQU's die Konstanten definieren oder die sich auf die Sprungtabelle der C'T KAT-Ce beziehen, sollten unbedingt am Programmanfang stehen.

```
Beispiel:   NAME1     EQU     $1234
            NAME2     EQU     $100
            NEUNAME   EQU     NAME1-NAME2+100
```

NEUNAME bekommt also den Wert $1398 zugeordnet.

ORG — In der ORG-Anweisung wird die Startadresse und danach, durch ein Komma getrennt, der Dateiname des Object-Codes angegeben. Es ist nicht zulässig die ORG-Anweisung mit einem Label zu versehen.

```
Beispiel:   ORG$1000,#4:CODENAME.CODE
```

END — beendet das zu übersetzende Programm. Eventuell noch folgender Text wird ignoriert.

LIST — Die LIST-Anweisung veranlaßt die Ausgabe eines Übersetzungsprotokolls. Es enthält von links nach rechts: Die Speicheradresse und den Objectcode, daran anschließend den Quelltext in formatierter Darstellung. Am Ende der Übersetzung wird eine unsortierte Referenztabelle ausgegeben. In ihr wird angezeigt, welche Werte die verschiedenen Label erhalten haben. LIST ist beim Start des Assemblers standardmäßig voreingestellt. Die Anweisung kann in einem Programm an beliebigen Stellen und beliebig oft gegeben werden. Sie brauchen also nicht immer das ganze Programm 'aufzulisten'. Der LIST-Befehl verlangsamt die Übersetzung nämlich erheblich.

NOLIST — beendet die Ausgabe des Übersetzungsprotokolls. Die Anweisung kann an beliebigen Stellen und beliebig oft gegeben werden.

LIST — P1 .. P4 veranlaßt die Ausgabe des Übersetzungsprotokolls über die Druckerschnittstelle der C'T KAT-Ce. Es ist zweckmäßig, den Drucker zuvor auf 132 Druckstellen einzustellen. Das folgende Prgrämmchen zeigt dies als Beispiel für einen EPSON-Drucker.

```
            ORG       $1000,#4:PR132.   CODE
PRINTINIT   EQU       $8001C0
PRINTOUT    EQU       $8001C8

            JSR       PRINTINIT         ;Schnittst. init.
            LEA       TABELLE,A1        ;Adr.Byte-Tab ➧ A1
LOOP        MOVE.B    (A1)+,D0          ;next Byte ➧ D0
            BEQ       FERTIG            ;Tab zu ende?
            JSR       PRINTOUT          ;Byte ➧ Drucker
            BRA       LOOP              ;➧ next Byte

FERTIG      RTS                         ;➧ Hauptmenü

TABELLE     DC.B      $0F               ;132 Zeich./Zeile
            DC.B      $1B,'R',3         ;Deut. Zeichensatz
            DC.B      0                 ;Tabellenende
```

WAIT hält den Übersetzungsvorgang an und wartet auf einen Tastendruck.

ASC übersetzt die Zeichenfolge zwischen den Apostrophen in ASCII-Code und fügt ihn in den Object-Code ein.
Beispiel:

```
NAME    ASC     'BEISPIEL'
```

ergibt den Object-Code in HEX: 4245 4953 5049 454C

STR übersetzt die Zeichenfolge zwischen den Apostrophen in ASCII-Code und speichert ihn mit der Länge vorweg in den Object-Code.
Beispiel:

```
NAME    STR     'BEISPIEL'
```

ergibt den OBJECT-Code in HEX: 0742 4549 5350 4945 4C

DC Define Code erzeugt je nach Zusatz eine Folge von Bytes, Worten oder Doppelworten. Die einzelnen Werte werden durch Kommata voneinander getrennt.
Beispiel:

```
NAME1   EQU     $AB
        DC.B    '1',45,$12,NAME1
```

ergibt den Object-Code in HEX: 312D 12AB
Hinweis:

```
        DC.B    'KATCE'
```

erzeugt nur ein Byte mit dem Inhalt $45 (letztes Zeichen von 'KATCE').

```
NAME1   EQU     $1234
NAME2   EQU     $0234
        DC.W    $ABCD,NAME1-NAME2
```

ergibt den Object-Code in HEX: ABCD 1000

```
        DC.L    0,0,0
```

ergibt im Object-Code 3 Doppelworte (12 Byte) mit dem Inhalt Null.

SYNC Die Struktur des M68000 erfordert es, daß Befehle und Wort-Daten an einer Wortgrenze, also geraden Adressen, beginnen. Anderenfalls kann die CPU ein Wort nicht mit einem Zugriff aus dem RAM lesen. Normalerweise ist diese Ausrichtung in einem Programm von Anfang an gegeben, dß alle Befehle eine gerade Anzahl von Bytes belegen. Lediglich die Definition von Bytes mit ASC, STR oder DC.B kann diese Ordnung stören. Geschieht dies am Ende eines Programms, ist es bedeutungslos, folgen aber noch Befehle, so muß bei Bedarf ein Byte eingefügt werden. Die SYNC-Anweisung veranlaßt den Assembler zur Überprüfung seines Adresszählers, falls er ungerade ist, wird ein Null-Byte eingefügt und die Übersetzung fortgesetzt. Fehlt eine SYNC-Anweisung, verhütet eine Fehlermeldung den möglichen Programmabsturz.

CHAIN veranlaßt den Assembler, die nächste Textportion von der Diskette zu lesen. Das Laufwerk und der Dateiname sind entsprechend den Anforderungen des Betriebssystems im Terminalrechner anzugeben.
Beispiel: `CHAIN  #5:NEUER.TEXT`

(* *) Mit den 'Befehlen' (* und *) können Sie größere Textpassagen zum Kommentar erklären. Die damit eingerahmten Programmteile werden vom Übersetzer ignoriert. Formal werden diese Zeichen wie Befehle behandelt und müssen deshalb auch deren Konventionen genügen.
Beispiel:

```
(* Dies ist ein
mehrzeiliger Kommentar
*)
```

In der nachfolgenden Übersicht sind alle Befehle aufgeführt, die der Assembler versteht. Es wurden dabei folgende Abkürzungen verwandt:

An	Adressregister n
D8	8-Bit-Adressdistanz
D16	16-Bit-Adressdistanz
Dn	Datenregister n
EA	effektive Adresse
PC	Programmzähler
SR	Statusregister
()	Inhalt von
‹›	obligatorische Angaben
-›	ergibt

ABCD — Add Binary Coded Decimal / Byte
ABCD Dy,Dx oder ABCD -(Ay),-(Ax)

ADD — Add Binary / Byte, Wort, Doppelwort
ADD ‹EA›,DN oder ADD Dn,‹EA›

ADDA — Add Adress / Wort, Doppelwort
ADD ‹EA›,An

ADDI — Add Immediate / Byte, Wort, Doppelwort
ADDI #‹Daten›,‹EA›

ADDQ — Add Quick / Byte, Wort, Doppelwort
ADDQ #‹Daten›;‹EA›; Daten nur 0 bis 7

ADDX — Add with Extend / Byte, Wort, Doppelwort
ADDX Dy,Dx oder ADDX -(Ay),-(Ax)

AND — And logical / Byte, Wort, Doppelwort
AND ‹EA›,Dn oder AND Dn,‹EA›

ANDI — And Immediate / Byte, Wort, Doppelwort
ANDI #‹Daten›,‹EA›

ANDI — And Immediate to Condition Code Register / Byte
ANDI #‹8-Bit-Daten›,CCR

ANDI — And Immediate to Status Register / Wort
ANDI #‹16-Bit-Daten›,SR

ASL — Arithmetic Shift Left / Byte, Wort, Doppelwort
ASL Dx,Dy oder ASL #‹Daten›,Dn oder ASL ‹EA›

ASR — Arithmetic Shift Right / Byte, Wort, Doppelwort
ASR Dx,Dy oder ASR #‹Daten›,Dn oder ASR ‹EA›

BCC — Branch Carry Clear
BCC ‹Label›

BCS — Branch Carry Set
BCS ‹Label›

BEQ — Branch Equal
BEQ ‹Label›

BNE	Branch Not Equal BNE ‹Label›
BGE	Branch Greater or Equal BGE ‹Label›
BGT	Branch Greater Than BGT ‹Label›
BHI	Branch Hi BHI ‹Label›
BLE	Branch Less or Equal BLE ‹Label›
BLS	Branch Less or Same BLS ‹Label›
BLT	Branch Less Than BLT ‹Label›
BMI	Branch Minus BMI ‹Label›
BPL	Branch Plus BPL ‹Label›
BVC	Branch Overflow Clear BVC ‹Label›
BVS	Branch Overflow Set BVS ‹Label›
BT	Branch True BT ‹Label›
BF	Branch False BF ‹Label›
BCHG	Bit Test and Change / Byte, Doppelwort BCHG Dn,‹EA› oder BCHG #‹Daten›,‹EA›
BCLR	Bit Test and Change / Byte, Doppelwort BCLR Dn,‹EA› oder BCLR #‹Daten›,‹EA›
BRA	Branch Always BRA ‹Label›
BSET	Bit Test and Set / Byte, Doppelwort BSET Dn,‹EA› oder BSET #‹Daten›,‹EA›
BSR	Branch to Subroutine BSR ‹Label›
BTST	Bit Test /Byte, Doppelwort BTST Dn,‹EA› oder BTST #‹Daten›,‹EA›
CHK	Check Register Against Bounds / Wort CHK ‹EA›,Dn
CLR	Clear Operand / Byte, Wort, Doppelwort CLR ‹EA›
CMP	Compare / Byte, Wort, Doppelwort CMP ‹EA›,Dn
CMPA	Compare Adress / Wort, Doppelwort CMPA ‹EA›,An

CMPI	Compare Immediate / Byte, Wort, Doppelwort CMPI #‹Daten›,‹EA›
CMPM	Compare Memory / Byte, Wort, Doppelwort CMPM (Ay)+,(Ax)+
DBCC	Test, Decrement and Branch Carry Clear / Wort DBCC Dn,‹Label›
DBCS	Test, Decrement and Branch Carry Set / Wort DBCS Dn,‹Label›
DBEQ	Test, Decrement and Branch Equal / Wort DBEQ Dn,‹Label›
DBNE	Test, Decrement and Branch Not Equal / Wort DBNE Dn,‹Label›
DBGE	Test, Decrement and Branch Greater or Equal / Wort DBGE Dn,‹Label›
DBGT	Test, Decrement and Branch Greater Than / Wort DBGT Dn,‹Label›
DBHI	Test, Decrement and Branch High / Wort DBHI Dn,‹Label›
DBLE	Test, Decrement and Branch Less or Equal / Wort DBLE Dn,‹Label›
DBLS	Test, Decrement and Branch Less or Same / Wort DBLS Dn,‹Label›
DBLT	Test, Decrement and Branch Less Than / Wort DBLT Dn,‹Label›
DBMI	Test, Decrement and Branch Minus / Wort DBMI Dn,‹Label›
DBPL	Test, Decrement and Branch Plus / Wort DBPL Dn,‹Label›
DBVC	Test, Decrement and Branch Overflow Clear / Wort DBVC Dn,‹Label›
DBVS	Test, Decrement and Branch Overflow Set / Wort DBVS Dn,‹Label›
DBT	Test, Decrement and Branch True / Wort DBT Dn,‹Label›
DBF	Test, Decrement and Branch False / Wort DBF Dn,‹Label›
DIVS	Signed Divide / Wort DIVS ‹EA›,Dn
DIVU	Unsigned Divide / Wort DIVU ‹EA›,Dn
EOR	Exclusive Or Logical/ Byte, Wort, Doppelwort EOR Dn,‹EA›
EORI	Exclusive Or Immediate / Byte, Wort, Doppelwort EORI #‹Daten›,‹EA›
EORI	Exclusive Or Immediate to CCR / Byte EORI #‹8-Bit-Daten›,CCR
EORI	Exclusive Or Immediate to SR / Wort EORI #‹16-Bit-Daten›,SR

EXG	Exchange Registers / Doppelwort EXG Rx,Ry
EXT	Sign Extend / Wort, Doppelwort EXT Dn
JMP	Jump JMP ‹EA›
JSR	Jump to Subroutine JSR ‹EA›
LEA	Load effective Adress / Doppelwort LEA ‹EA›,An
LINK	Link and Allocate LINK An,#‹D16›
LSL	Logical Shift Left / Byte, Wort, Doppelwort LSL Dx,Dy oder LSL #‹Daten›,Dn oder LSL ‹EA›
LSR	Logical Shift Right / Byte, Wort, Doppelwort LSR Dx,Dy oder LSR #‹Daten›,Dn oder LSR ‹EA›
MOVE	Move Data / Byte, Wort, Doppelwort MOVE ‹EA›,‹EA›
MOVE	Move Data to CCR / Wort MOVE ‹EA›,CCR
MOVE	Move Data to SR / Wort MOVE ‹EA›,SR
MOVE	Move from SR / Wort MOVE SR,‹EA›
MOVE	Move User Stack Pointer / Doppelwort MOVE USP,An oder MOVE An,USP
MOVEA	Move Adress / Wort, Doppelwort MOVEA ‹EA›,An
MOVEM	Move Multiple Registers / Wort, Doppelwort MOVEM ‹Reg.Liste›,‹EA› oder MOVEM ‹EA›,‹Reg.Liste› Beispiele für ‹Reg.Liste›: D5-A2 bezeichnet die Register D5,D6,D7,A0,A1 und A2 D0,D1,D4/D6-A0 bezeichnet die Register D0,D1,D4,D6,D7 und A0
MOVEP	Move Peripheral Data / Wort, Doppelwort MOVE Dx,D(Ay) oder MOVE D(Ay),Dx
MOVEQ	Move Quick / Doppelwort MOVEQ #‹8-Bit-Daten›,Dn
MULS	Signed Multiply / Wort MULS ‹EA›,Dn
MULU	Unsigned Multiply / Wort MULU ‹EA›,Dn
NBCD	Negate Decimal with Extend / Byte NBCD ‹EA›
NEG	Negate / Byte, Wort, Doppelwort NEG ‹EA›

NEGX	Negate with Extend / Byte, Wort, Doppelwort NEGX ‹EA›
NOP	No Operation NOP
NOT	Logical Complement / Byte, Wort, Doppelwort NOT ‹EA›
OR	Logical Or / Byte, Wort, Doppelwort OR ‹EA›,Dn oder OR Dn,‹EA›
ORI	Logical Or Immediate / Byte, Wort, Doppelwort ORI #‹Daten›,‹EA›
ORI	Logical Or Immediate to CCR / Byte ORI #‹8-Bit-Daten›,CCR
ORI	Logical Or Immediate to SR / Wort ORI #‹16-Bit-Daten›,SR
PEA	Push Effective Adress / Doppelwort PEA ‹EA›
RESET	Reset External Devices RESET
ROL	Rotate Left without Extend / Byte, Wort, Doppelwort ROL Dx,Dy oder ROL #‹Daten›,Dn oder ROL ‹EA›
ROR	Rotate Right without Extend / Byte, Wort, Doppelwort ROL Dx,Dy oder ROL #‹Daten›,Dn oder ROL ‹EA›
ROXL	Rotate Left with Extend / Byte, Wort, Doppelwort ROXL Dx,Dy oder ROXL #‹Daten›,Dn oder ROXL ‹EA›
ROXR	Rotate Right with Extend / Byte, Wort, Doppelwort ROXR Dx,Dy oder ROXR #‹Daten›,Dn oder ROXR ›EA›
RTE	Return from Exception RTE
RTR	Return and Restore Condition Codes RTR
RTS	Return from Subroutine RTS
SBCD	Subtract Decimal with Extend / Byte SBCD Dy,Dx oder SBCD -(Ay),-(Ax)
SCC	Set according to Condition Carry Clear / Byte SCC ‹EA›
SCS	Set according to Condition Carry Set / Byte SCS ‹EA›
SEQ	Set according to Condition Equal / Byte SEQ ‹EA›
SNE	Set according to Condition Not Equal / Byte SNE ‹EA›
SGE	Set according to Condition Greater or Equal / Byte SGE ‹EA›
SGT	Set according to Condition Greater Than / Byte SGT ‹EA›
SHI	Set according to Condition High / Byte SHI ‹EA›

SLE	Set according to Condition Less or Equal / Byte SLE ‹EA›
SLS	Set according to Condition Less or Same / Byte SLS ‹EA›
SLT	Set according to Condition Less Than / Byte SLT ‹EA›
SMI	Set according to Condition Minus / Byte SMI ‹EA›
SPL	Set according to Condition Plus / Byte SPL ‹EA›
SVC	Set according to Condition Overflow Clear / Byte SVC ‹EA›
SVS	Set according to Condition Overflow Set / Byte SVS ‹EA›
ST	Set according to Condition True / Byte ST ‹EA›
SF	Set according to Condition False / Byte SF ‹EA›
STOP	Load Statusregister and Stop STOP ‹16-Bit-Daten›
SUB	Subtract Binary / Byte, Wort, Doppelwort SUB ‹EA›,Dn oder SUB Dn,‹EA›
SUBA	Subtract Adress / Wort, Doppelwort SUBA ‹EA›,An
SUBI	Subtract Immediate / Byte, Wort, Doppelwort SUBI #‹DAten›,‹EA›
SUBQ	Subtract Quick / Byte, Wort, Doppelwort SUBQ #‹Daten›,‹EA› ; Daten 1 bis 8
SUBX	Subtract with Extend / Byte, Wort, Doppelwort SUBX Dy,Dx oder SUBX -(Ay),-(Ax)
SWAP	Swap Register Halves / Wort SWAP Dn
TAS	Test and Set Operand / Byte TAS ‹EA›
TRAP	Trap TRAP #‹Daten›; Daten 0 bis 15
TRAPV	Trap on Overflow TRAPV
TST	Test an Operand / Byte, Wort, Doppelwort TST ‹EA›
UNLK	Unlink UNLK An

Für weitergehende Informationen über die Wirkungsweise der einzelnen Befehle müssen wir auf die im Handel angebotene Literatur verweisen.

4.4.2 Bedingtes Übersetzen / conditional assembly

Oft ist es erforderlich, von einem Programm mehrere Versionen bereitzuhalten und zu pflegen. Dies kann bei großen Programmen und vielen Versionen ganz schön aufwendig und fehleranfällig werden. In diesem Fall kann das 'conditional assembly' weiterhelfen. Hiermit ist es möglich, mit Hilfe von Bedingungen ganze Passagen des Quelltextes von der Übersetzung auszuschließen oder gezielt mit einzubeziehen.

IF — Der Operand dieser Anweisung wird während der Übersetzung vom Assembler getestet. Ist er gleich Null, werden die nachfolgenden Quelltext-Zeilen bis ELSE- oder ENDIF-Anweisung überlesen bzw. wie Kommentar behandelt. Ist der Operand ungleich Null, werden die folgenden Zeilen übersetzt. Als Operand ist jeder Ausdruck zulässig.

ELSE — Die hiernach folgenden Zeilen bis zur ENDIF-Anweisung werden übersetzt, wenn der Operand zur IF-Anweisung gleich Null ist. Anderenfalls werden sie überlesen bzw. wie Kommentar behandelt.

ENDIF — beendet die 'bedingte Übersetzung. Die jetzt folgenden Zeilen werden unbedingt übersetzt.

Die Anweisungfolge IF, ELSE und ENDIF kann in einem Programm beliebig oft wiederholt werden. Die ELSE-Anweisung ist nur notwendig, wenn die Logik dies erfordert.
Beispiel:

```
TESTWERT  EQU     0
          IF      TESTWERT
LADED0    MOVE.W  $1234,D0
          ELSE
LADED0    MOVE.W  $5678,D0
          ENDIF
```

Im Object-Code wird der Befehl (es dürfen auch viele sein) MOVE.W $5678,D0 abgesetzt. Wie das Beispiel zeigt, sind auch Label-Definitionen erlaubt.

4.4.3 RAM-Belegung beim Arbeiten mit Editor und Assembler

Abhängig von der Größe Ihres Einkommens können Sie den RAM der C'T KAT-Ce unterschiedlich bestücken. In der Minimalversion enthält die Platine 4 RAM-IC's von je 8 Kilobyte. Insgesamt also 32K. Dieser Bereich wird vom Monitor in Text- und Codespeicher aufgeteilt. Das Ergebnis dieser Bemühungen wird im Hauptmenü angezeigt. Sollte Ihnen diese Aufteilung nicht gefallen, können die entsprechenden Pointer vom Monitor aus vorgegeben werden. Die Adressen sind:

TEXTSTART = $3B4
RAMENDE = $3B8 (Textende)
CODESTART = $3BC

Für den Text stehen in der Minimalversion ca. 20K zur Verfügung (mehr hat APPLE-Pascal auch nicht). Reicht dies nicht aus, müssen Sie den Text in mehrere Dateien aufteilen und diese dann zum Assemblieren aneinanderketten. Wie das geht, finden Sie bei den Assembleranweisungen unter 'CHAIN'. Der Object-Code wird dann mit Hilfe einer ORG-Anweisung auf der Diskette abgelegt.

Während im ersten Fall der Assembler den Codespeicher zwischen Code und Label nach Bedarf dynamisch aufgeteilt, muß im zweiten Fall mit einem reservierten Labelspeicher gearbeitet werden. Immerhin darf das Programm dann 675 Label enthalten.

4.5 Datei-Organisation

Files werden immer blockweise bearbeitet. Die Blocklänge beträgt 512 Bytes. Zu jedem File gehört eine Filenummer, die bei allen Diskettenoperationen anzugeben ist. Lediglich IoResult ist von einer Filenummer unabhängig.
Der Diskettenzugriff mit Hilfe von Filenummern erleichtert das Arbeiten mit mehreren gleichzeitig eröffneten Files. Der Assembler der C'T KAT-Ce z.B. nutzt bei der Disketten-Assemblierung gleichzeitig zwei Files (Filenummer 0 und 1).
Die Anzahl der Files, die gleichzeitig eröffnet sein können, wird nur durch das Betriebsystem des Terminalrechners begrenzt.
Der Filename wird von der C'T KAT-Ce als String mit der Länge vorweg gesendet.
Es kann davon ausgegangen werden, daß die C'T KAT-Ce die Daten wesentlich schneller sendet, als der Terminalrechner sie verarbeiten kann. Damit keine Daten verloren gehen, muß das Terminalprogramm während länger andauernder Operationen die RTS-Leitung sperren und danach wieder frei geben. Dies gilt insbesondere für Diskettenoperationen und Bildschirmausgaben.
Innerhalb der 512-Byte-Blöcke speichert der Editor die Texte inZeilen variabler Länge. Jede Zeile beginnt mit dem Steuerzeichen $10. Es schließt sich $20+n an, wobei n die Anzahl der Blanks am Zeilenanfang darstellt. Danach folgt der Zeileninhalt in ASCII-Darstellung, endend mit $0D. Das Textende wird mit $00 markiert.
Die Dateiorganisation von APPLE-PRODOS weist eine bemerkenswerte Eigenart auf. PRODOS gibt den Files eigene Nummern, die unabhängig von den Filenummern der C'T KAT-Ce sind. Ein Schließaufruf (Close) bewirkt nun, daß nicht nur das angegebene File geschlossen wird, sondern auch alle anderen, danach eröffneten Files (mit einer größeren Filenummer).

4.6 Das Terminal-Programm

Für die in 2.2 bis 2.5 genannten Rechner können Sie von C'T auf Anforderung ein geeignetes Terminal-Programm erhalten. Falls Sie C'T KAT-Ce an einen anderen Rechner anschließen möchten, oder Ihnen das Standard-Proramm nicht gefällt, müssen Sie selbst ein Programm schreiben. Zur Unterstützung finden Sie nachfolgend einen Programmablaufplan in tabellarischer Form.

START
1.Initialisierung der Schnittstelle
Bei der Bestellung der C'T KAT-Ce können Sie wählen, ob die Verbindung zwischen Terminal-Rechner und C'T KAT-Ce über eine serielle oder eine parallele Schnittstelle erfolgen soll. Die ROMs der C'T KAT-Ce enthalten dann das jeweils richtige Treiberprogramm.

a) parallele Schnittstelle:
Sobald die C'T KAT-Ce sich empfangsbereit gemeldet hat, sendet das Terminalprogramm ein ‹CR› und springt zur Hauptroutine (READKATCE).
b) serielle Schnittstelle:
Zur automatischen Erkennung der Baudrate liest die C'T KAT-Ce solange Zeichen von der Schnittstelle, bis ein ‹CR› richtig erkannt wird. Aus diesem Grund muß die Initialisierung der Schnittstelle in einer Schleife erfolgen.

```
SERINITSTART    Sende ein ‹CR› zur C'T KAT-Ce
                0 ⧫ ZAEHLER
LOOP            warte 1/10000 Sekunde
                ‹CR› angekommen?
                    ja: ⧫ READKATCE   ; Initialisierung fertig
                  nein: ZAEHLER +1
                        ZAEHLER = 1000?
                          ja: ⧫ SERINITSTART
                        nein: ⧫ LOOP
```

2.Hauptroutine (identisch für für beide Schnittstellen)

```
READKATCE       lies nächstes Zeichen von der C'T KAT-Ce
                wenn Zeichen = $11: ⧫ CHAR11 ; Sonderbearbeitung
                wenn Zeichen = $12: ⧫ CHAR12 ; Keypressbearbeitung
                wenn Zeichen = $13: ⧫ CHAR13 ; Tastaturbearbeitung
                sonst: Zeichen auf Bildschirm ausgeben,
                ⧫ READKATCE
Tastaturbearbeitung
CHAR13          lies nächstes Zeichen von der Tastatur.
                (Wenn noch kein Zeichen eingegeben wurde, muß auf die
                Eingabe gewartet werden)
                sende Tastenwert ⧫ C'T KAT-Ce
                ⧫ READKATCE
Keypressbearbeitung
CHAR12          ist oder wurde eine Taste gedrückt?
                  ja: sende $01 ⧫ C'T KAT-Ce, ⧫ READKATCE
                nein: sende $00 ⧫ C'T KAT-Ce, ⧫ READKATCE
Sonderbearbeitung
CHAR11          lies nächstes Byte von der C'T KAT-Ce
                wenn Byte = $00: ; Setup
                    Sende maximale X- und Y-Koordinate
                    zur C'T KAT-Ce.
                    ⧫ READKATCE
                wenn Byte = $01: ; UnitOut
                    lies nächstes Byte von der C'T KAT-Ce
                    und sende es zum Drucker.
                    ⧫ READKATCE
                wenn Byte = $02: ; Unitinit
                    initialisiere Druckerschnittstelle.
                    ⧫ READKATCE
                wenn Byte = $03: ; BlockRead
                    lies 1 Byte Filenummer und 2 Bytes Blocknummer
```

(HighByte first) von der C'T KAT-Ce.
lies den dadurch spezifizierten Block (512 Bytes)
von der Diskette und sende ihn an die C'T KAT-Ce.
♦READKATCE

wenn Byte = $04: ; BlockWrite
lies 1 Byte Filenummer und 2 Bytes Blocknummer
(HighByte first) von der C'T KAT-Ce,
lies 512 Bytes von der C'T KAT-Ce und schreibe den
Block an die Filenummer/Blocknummer
entsprechende Stelle auf der Diskette.
♦READKATCE

wenn Byte = $05: ; Resetfile
lies von der C'T KAT-Ce 1 Byte Filenummer,
1 Byte N und N-Bytes Filename.
Öffne File(Filename) zum Lesen.
♦READKATCE

wenn Byte = $06: ; RewriteFile
lies von der C'T KAT-Ce 1 Byte Filenummer,
1 Byte N und N-Bytes Filename.
Öffne File (Filename) zum Schreiben.
♦READKATCE

wenn Byte = $07: ; CloseFile
lies von der C'T KAT-Ce 1 Byte Filenummer
und schließe das zugehörige File.
♦READKATCE

wenn Byte = $08: ; CloseDeleteFile
lies von der C'T KAT-Ce 1 Byte Filenummer
und lösche das zugehörige File.
♦READKATCE

wenn Byte = ♦09: ; IoResult
sende das IoResult-Byte der letzten Disketten-
Operation an die C'T KAT-Ce.
♦READKATCE

wenn Byte = $0A: ; GotoXY
lies von der C'T KAT-Ce das X-Koord-Byte und das
Y-Koord-Byte. Setze den Cursor an diese Stelle des
Bildschirms. X=0,Y=0 entspricht der oberen linken
Ecke des Schirms.
♦READKATCE

wenn Byte = $0B: ; Quit
beende das Terminalprogramm.

Weitere Sondercodes folgen event. später. Der Bereich ab $80 steht für Eigenentwicklungen zur Verfügung.

5.1 Funktionsbeschreibung und Adressdekodierung

Die MC68000 CPU kann mit ihren 23 Adressleitungen (A1 bis A23) 8 Megaworte zu je 16 Bit adressieren. Die Adressleitung A0 gibt es nicht, ihre Funktion wird durch die Leitungen UDS und UDL übernommen. Diese geben an, ob bei einem Speicherzugriff das höherwertige (UDS= Upper Data Strobe) und/oder das niederwertige (LDS=Lower Data Strobe) Datenbyte eines Wortes zu lesen oder zu schreiben ist. Mit einem 2-Bit Binärdekoder (1/2 74LS139), von dessen Ausgängen Q0, Q1, Q2 und Q3 immer nur einer aktiv ist, wird dieser Adressraum in vier gleichgroße Bereiche zu je 4 MB aufgeteilt.

Der unterste Bereich ($000000 bis $3FFFFF) beinhaltet, je nach Stellung des RS-Flipflops (2/3 74LS10), die ROMs oder die RAMs. Je nach Lage dieses Flipflops schaltet ein Multiplexer (1/2 74LS157) das Speicherzugriffs-Signal aus Q0 (Pin 12 des 74LS139) entweder zu den ROMs oder zu den RAMs durch. Nach dem Einschalten und nach jedem Reset werden zunächst die ROMs und dann die RAMs in diesen Bereich eingeblendet. Die Umschaltung erfolgt, nachdem die CPU aus $0 bis $3 den Stackpointer und danach aus $4 die Startadresse gelesen hat. Dadie Startadresse auf den Bereich ab $800000 zeigt, kann die Umschaltung mit dem o.g. Signal aus Q0 erfolgen. Weil die CPU die Vektoren weiterhin ab Adresse S0 erwartet, müssen diese nach jedem Reset aus den ROMs in die RAMs übertragen werden. Die weitere Auswahl der RAMs ist etwas kompliziert, weil RAM-Bausteine unterschiedlicher Größe in zwei Bänken lückenlos hintereinander zu adressieren sind. Die Lösung besteht darin, daß das Chip-Select-Signal für die RAMs je nach Type entweder aus A14 oder aus A16 hergeleitet wird. Für die Verknüpfung mit UDS/LDS sorgt ein 74LS139-System und ein 74LS32. Da auch die Pin-Belegung der beiden RAM-Typen nicht identisch ist, wurden für die notwendigen Umschaltungen zwei Jumper installiert.
Diese Form der Umschaltung legt an alle RAMs die gleichen Adress-Signale an und hat zur Folge, daß bei gemischter Bestückung die 'kleinen' RAMs mehrfach abgebildet werden. Die mitgelieferte Software kann dies nicht berücksichtigen und eine gemischte Bestückung ist deshalb bei Einsatz der Standardsoftware nicht sinnvoll.
Die Adressleitungen A15 bis A21 bzw. A17 bis A21 werden nicht ausgewertet und die Inhalte der RAMs sind deshalb auch unter anderen Adressen lesbar und zwar sooft, wie die als Binärzahl interpretierten unbenutzten Adressbits angeben.
Da die RAMs für eine mit 8 Mhz betriebene CPU schnell genug sind, braucht kein Wartezyklus erzeugt zu werden. Die Quittierung des RAM-Zugriffes (DTACK) erfolgt deshalb, über ein ein 74LS10-Gatter, dessen Ausgang zusätzlich invertiert wird, ebenfalls mit dem Signal von Q0.

Der zweite Bereich ($400000 bis $7FFFFF), selektiert durch Q1 (Pin 11 des 74LS139), wird mit einem zweiten 74LS139-System durch A13 und LDS noch einmal unterteilt. Da A14 bis A21 unberücksichtigt bleiben, wird ein 8K Worte großer Bereich in $400000 bis $7FFFFF 256 mal abgebildet. Wegen der Berücksichtigung von LDS, ist immer nur das niederwertige Byte eines Wortes selektiert. A13 teilt diesen 8KW-Bereich in zwei gleichgroße Teile.
Der untere Teil ist dem Port-Baustein MC68230 zugeordnet. Dieses IC belegt mit seinen Eingängen RS1 bis RS5 die Adressleitungen A1 bis A5. Die Leitungen A6 bis A12 und A14 bis A21 sind also unbenutzt. Daraus folgt, daß die Register des MC68230 ab $400001 alle 8KW 128 mal selektiert werden. Die drei Ports dieses Bausteins sind direkt

an entsprechende Pfostenfeld-Stifte amRand der Platine geführt.

Der obere Teil ist dem DUART (Dual Asynchronous Receiver/Transmitter) zugeteilt. Dieses IC belegt die Adressleitungen A1 bis A4 und wird deshalb ab $402001 alle 8KW 256 mal eingeblendet. Der Baustein enthält zwei Sende/Empfänger für den seriellen Datenverkehr. Dadieser in der Regel über +/- 12 Volt-Pegel erfolgt, sind entsprechende Treiber (1488/1489) vorgesehen. Den Takt, für die per Software einstellbaren Baudraten, erzeugt der im IC enthaltene Taktgenerator mit einem 3.6864 MHz Quarz.
Beide Bausteine, MC68230 und MC68681, erzeugen selbst ein DTACK-Signal.

Der dritte Bereich ($800000 bis $BFFFFF) enthält nach der Umschaltung die ROMs.Im Falle der standardmäßig eingesetzten ROMs (27128) sind die Adressleitungen A1 bis A14 belegt und die Adressleitungen A15 bis A21 bleiben unberücksichtigt. Daraus folgt, daß die ROMs 128 mal (2EX6) hintereinander im Bereich $800000 bis $BFFFFF lesbar sind.

Der höchste Bereich ($C00000 bis $FFFFFF) ist dem Digital-Analog-Wandler, dem Analog-Digital-Wandler und der Expansionsschnittstelle zugewiesen. Ein weiteres 74LS139-System teilt diesen Bereich mit A14 und A15 in vier 8KW-Blöcke. Die Selektierung erfolgt über Q3 (Pin 9 vom 74LS139), synchronisiert durch VPA/VMA. Weil hier die Adressleitungen A16 bis A21 unberücksichtigt bleiben, wiederholen sich die vier Blöcke, beginnend bei $C00000, im 32KW-Raster 64 mal.
Der erste 8KW-Block bedient die Expansionsschnittstelle. An ihr sind, neben einigen Steuersignalen, das niederwertige Datenbyte und die Adressleitungen A1 bis A13 herausgeführt. Es können also Datenbytes an 4096 verschiedenen ungeraden Adressen ein- bzw. ausgegeben werden. Da die Schnittstelle direkt am Daten- und Adressbus der C'T KAT-Ce anliegt, muß hier bei Bedarf mit geeigneten ICs (z.B. 74LS244 oder 74LS245) angekoppelt werden. Ein negativer Impuls an Pin 8 (Select invertiert) zeigt an, daß die Daten, Adressen und übrigen Signale für die Schnittstelle bestimmt sind.

Schreiben in die Speicherstelle $C04000 (und alle weiteren bis $C07FFE) versorgt den Digital-Analog-Wandler ZN428 mit einem neuen Digitalwert.

Lesen aus der Speicherstelle $C0C000 startet den Analog-Digital-Wandler ZN427. Der digitalisierte Analogwert kann 10 Mikrosekunden später aus $C08000 gelesen werden. Auch diese Adressen wiederholen sich 8192 mal hintereinander.

Ein DTACK-Signal braucht für den Bereich $C00000 bis $FFFFFF nicht erzeugt zu werden. Es wird von der CPU nach einem onen Zugriff nicht erwartet.

Taktgenerator und Resetsignalerzeugung weisen keine Besonderheiten auf.

5.2 Speicherchips und Jumper

Obwohl Sie mit der Platine zwei EPROMs mitgeliefert bekommen, können Sie bei Bedarf auch andere Typen verwenden. Die nachfolgende Tabelle gibt Aufschluß dar-

über, welche Jumper bei welchem EPROM-Typ zu stecken sind. Die Angaben beziehen sich auf das EPROM-Jumperfeld, das zwischen der CPU und dem High-Byte-EPROM angeordnet ist. Wenn die Platine mit der Bestückungsseite nach oben und dem V24-Stecker nach rechts, vor Ihnen liegt, ist der Steckplatz für Jumper 1 auf der linken Seite. Die Jumper sind so zu stecken, daß jeweils zwei übereinander liegende Stifte miteinander verbunden werden. Falls Sie 2732-EPROMs verwenden wollen: Diese müssen im betreffenden Sockel so gesteckt werden, daß die Pins 1,2 und 27,28 des Sockels frei bleiben.

EPROM-Typ	Jumper auf
2732 o. 2764	2 4 6
27128	1 4 6
27256	1 3 6
27512	1 3 5

Bei den RAMs können Sie zwei verschiedene Typen in vier Kombinationen verwenden. In der Minimal-Version kommen zwei 6264 mit je 8K Byte zum Einsatz und in der Maximal-Version werden vier 43256 mit je 32K Byte verwendet. Diese Minimal-Version ist bei den derzeitigen RAM-Preisen nicht mehr und die Maximal-Version noch nicht diskutabel. Dazwischen geht es noch mit 4 X 6264 oder 2 X 43256. Andere Kombinationen sind wegen der Wortstruktur der CPU nicht erlaubt. Wenn Sie nur zwei RAMs verwenden, müssen Sie diese in die beiden Sockel oberhalb der ROMs stecken. Die Tabelle gibt an, wie die Jumper zu stecken sind. Gemeint ist das Jumperfeld rechts vom High-Byte-EPROM. Jumpersteckplatz 1 befindet sich am oberen Ende des Jumperfeldes. Die Jumper müssen so gesteckt werden, daß nebeneinander liegende Sifte miteinander verbunden werden.

RAM-Typ/ Kombination	Jumper
2X6264	1 3
4X6264	1 3
2X43256	2 4
4X43256	2 4

Bitte beachten Sie, daß die CPU und der 68230 im Vergleich zu den anderen ICs auf der Platine um 180 Grad gedreht sind.

5.3 Hinweise für Selbstbauer

Es hat sich herausgestellt, daß die beiden Peripherie-ICs und der 3.6864 MHz-Quarz nicht überall zu bekommen sind. Überlegen Sie sich deshalb, ob es nicht besser ist einen kompletten Bausatz oder gar eine fertige C'T KAT-Ce zu bestellen. Aber wer einen guten Lötkolben hat, möchte diesen natürlich auch nutzen ....
Die Platine wird ohne Bestückungsdruck geliefert. Stattdessen erhalten Sie einen vergrößerten Bohrplan, auf dem alle Bauteile eingezeichnet sind. Dieses Verfahren scheint uns bei einer so fein gewobenen Platine sicherer als ein Bestückungsdruck, der oft genug ein bischen verschoben ist und dann im Zweifelsfall keine genaue Auskunft mehr gibt. Außerdem ist die Platine so voll gepackt, daß beim fertig aufgebauten Rechner vom Bestückungsdruck nicht mehr viel zu sehen wäre.

Für die Funktion der C'T KAT-Ce werden zumindest am Anfang nicht alle Bauteile benötigt. Insbesondere die DA- und AD-Chips können Sie auch noch später kaufen. Dennoch möchten wir Ihnen nahe legen, die Sockel und die übrigen Billigteile gleich mit einzulöten. Sie brauchen dann bei einer späteren Nachrüstung nur noch die ICs einzustöpseln und nicht erst den ganzen Rechner auseinander zu nehmen. Dies ist bei den verwendeten CMOS RAMs nämlich nicht ohne Risiko, und an der IC-bestückten Platine sollte keinesfalls gelötet werden.
Wenn Sie den 48-Pin-Sockel nicht bekommen können, es geht auch mit zwei anreihbaren a 24. Falls auch der 64er nicht erhältlich ist, können Sie einseitige Streifensockel verwenden oder 40er Sockel mit der Laubsäge zerlegen. Bitte löten Sie keinesfalls die ICs ohne Sockel ein, denn ein Fehler ist dann mit vertretbarem Aufwand nicht mehr zu finden. Falls Sie selbst den Rechner nicht zum Laufen bringen, helfen wir Ihnen gerne, aber nur wenn alle ICs gesockelt sind. Die Netzwerke können auch aus Einzelwiderständen aufgebaut werden. Der allen Widerständen gemeinsame Pol ist auf dem Plan mit einem kleinen Viereck gekennzeichnet.
Und nun zum Aufbau ...
Die Bestückungsseite können Sie leicht an den in Platinenlängsrichtung verlaufenden Leiterbahnen erkennen. Außerdem sind neben Pin 1 der CPU die Rechnerbezeichnung und die Versionsnummer zu lesen.
Löten Sie zuerst die Widerstände, dann die Sockel, Kondensatoren, Pfostenfeldstifte und V24-Buchse ein. Die Quarze lassen Sie am besten bis zum Schluß, sonst sind die Drähtchen schon abgebrochen, bevor der Rechner läuft.
Noch ein paar Feinheiten ...
Das 12-polige Jumperfeld für die Selektion der ROMs muß so eingelötet werden, daß die beiden Durchkontaktierungen neben Pin 38 der CPU frei bleiben.
Die Quarze sollten mit ihrem Gehäuse ein bischen Abstand zur Platine haben. Anderenfalls kann das Gehäuse ungewollte Verbindungen erzeugen. Es gibt für diesen Zweck spezielle Unterlegscheibchen.
Nahe neben Pin 37 der CPU ist eine Durchkontaktierung. Seien Sie hier besonders vorsichtig, damit keine ungewollte Verbindung entsteht.
Wenn die V24-Buchse ohne Befestigunswinkel eingesetzt wird, sichern Sie vorher die beiden Löcher der Buchse mit zwei kleinen Schrauben oder Nieten. Es ist vorgekommen, daß die beiden Blechteile auseinander gehen und die Buchse sich dann in ihre vielen Einzelteile zerlegt.
Sehen Sie zum Schluß alle Lötstellen noch einmal nach. Am besten mit der Lupe. Achten Sie auch auf vergessene Lötstellen, denn diese können besonders interessante Fehler erzeugen.
Für den ersten Test werden je nach Schnittstelle der 68681 oder der 68230, die beiden RAMs am oberen Platinenrand und die AD- bzw. DA-Wandler nicht benötigt. Im Sinne einer Risikominimierung sollten diese ICs erst gesteckt werden, wenn der Rechner grundsätzlich funktioniert.
Wenn Sie bis hierher alles sorgfältig erledigt haben, können Sie nun Abschnitt 2.1 lesen und dort weitermachen. Wir drücken Ihnen die Daumen.

5.4 Teile-Liste

ICs

1	MC68000	IC1
1	MC68230	IC2
1	MC68681	IC3
1	NE555	IC4
2	MC1488[1]	IC5,IC6
2	MC1489	IC7,IC8
1	74LS04	IC91
1	74LS10	IC10
1	74LS32	IC11
2	74LS139	IC12,IC13
1	74LS157	IC14
2	27128[2]	IC15,IC16
4	6264[3]	IC17-IC20

Widerstände:

1	3K3	R1
1	1M	R2
2	1K	R3,R4
2	390	R5,R7
1	180K	R6
1	Array 7 X 3K3	Netzwerk1
1	Array 4 X 10K	Netzwerk2 (kann entfallen)

Kondensatoren:

1	100nF	C1
1	6n8	C2
2	10uF 10V	C3,C6
2	10p	C4,C5
16	100nF	

IC-Sockel:

1	64 polig
1	48 polig
1	40 polig
6	28 polige
1	18 polig
2	16 polig
9	14 polige
1	8 polig

Pfostenfeldstecker
2-reihig:

1	34 Stifte

sonstiges:

1 Quarz 8 MHz
1 Quarz 3.6864 MHz
8 Jumper
10Lötnägel 1mm, event. Spezialkabel mit Buchse/Stecker
1 Reset-Taster, normal offen
25-polige SubD-Buchse

1 20 Stifte
1 14 Stifte
2 12 Stifte
3 10 Stifte
1 8 Stifte

1) event. 2 X 74LS00, siehe hierzu Abschnitt 2.1
2) im Bausatz enthalten, werden mit der Platine geliefert
3) verschiedene Bestückung möglich, siehe hierzu Abschnitt 5.2

5.5 ... läuft nicht, was nun?

Wenn Sie die C'T KAT-Ce fertig gekauft haben, kann der Fehler nur in der Umgebung des Rechners liegen. Bitte überprüfen Sie deshalb zuerst die Spannungsversorgung des Rechners. Die +5 Volt müssen zwischen +4,75 und 5,25 Volt liegen und auch der 'Brumm' muß sich innerhalb dieser Grenzen halten.

Soll die C'T KAT-Ce mit dem Terminalrechner über die serielle Schnittstelle korrespondieren, so muß wie in 2.1 beschrieben die Spannungsversorgung an den +-12 Volt Anschlüssen zum Typ des Sendebausteins (1488 oder 74LS00) passen. Falls dies in Ordnung ist, messen Sie noch die Spannung an Pin 20 der V24-Buchse. Hier muß, auch ohne angeschlossenes Kabel zum Terminal, eine Spannung größer 2,5 Volt messbar sein. Diese Spannung geht auf Low-Pegel (0 oder unter 0 Volt), wenn Sie die Rest-Taste gedrückt halten.
Wenn bis hierher alles richtig funktioniert, kann der Fehler nur noch in der Pinbelegung des V24-Steckers oder im Terminal liegen. Die Pinbelegung der Buchse ist in 5.6.4 beschrieben und in 5.6.1 näher erläutert. Ist am Terminal eine serielle Schnittstelle vorhanden, hat die Buchse im Normalfall die gleiche Belegung. Ein Zwischenstück muß dann für die richtige Pin-Pin-Verbindung sorgen. Bewährt hat sich folgende Schaltung:

Pin	Pin	
2	mit 3	TxD mit RxD
3	mit 2	RxD mit TxD
4	mit 5	RTS mit CTS
5	mit 4	CTS mit RTS
7	mit 7	GND
8	mit 20	DCD mit DTR
20	mit 8	DTR mit DCD

Eine Ausnahme bildet das in 2.2 gezeigte PRODOS-Kabel für APPLE-Anwender. Falls Sie damit arbeiten, benötigen Sie kein Zwischenstück.
Läuft immer noch nicht?
Wenn Sie als Selbstbauer die obigen Zeilen gelesen haben und die Spannung an Pin 20 der V24-Buchse geht mit der Reset-Taste auf und ab, so funktioniert die C'T KAT-Ce sicher. Die Initialisierung der Schnittstelle erfolgt nämlich per Software und bis dahin ist schon eine ganze Menge gelaufen. Sollte sich an Pin 20 nichts bewegen, müssen Sie etwas tiefer einsteigen.
Prüfen Sie zunächst die Stromaufnahme der unbestückten Karte. Nach dem Aufladen der Kondensatoren darf kein nennenswerter Strom mehr fließen. Setzen Sie dann die

TTL-ICs ein und messen Sie an Pin 8 des 74LS04 die Taktfrequenz. Wenn Sie nur ein Multimeter besitzen: nehmen Sie einen 1K Widerstand in die Tastspitze und messen darüber die Spannung. Der gemessene Wert sollte bei 2 Volt liegen. Auf dem Oszillographen zeigen sich 8 MHz mit Tastverhältnis 1:1. Pin 10 des 74LS04 treibt das Reset-Signal. Es muß ca. 4 Volt betragen und bei gedrückter Reset-Taste auf 0 Volt gehen. Setzen Sie nun die CPU, die ROMs und RAMs und je nach Schnittstelle zum Terminal den 68681 (seriell) oder den 68230 (parallel) ein. Achten Sie bei den ROMs auf H und L und seien Sie beim Stecken der Jumper besonders sorgfältig. Wenn der Rechner nun nicht läuft und Sie nur ein Multimeter besitzen, bleibt auch Ihnen nur die Inanspruchnahme fremder Hilfe.
Mit dem Oszillographen läßt sich noch der ganze Rechner durchmessen. Anhand von 5.1 können Sie mit etwas Background die Arbeitsweise verstehen.
Prüfen Sie zunächst das CS-Signal an Pin 35 des 68681 oder Pin 41 des 68230. Je nach gewünschter Schnittstelle muß einer dieser Bausteine alle 3 bis 4 Mikrosekunden abgefragt werden. Ist dort nichts vorhanden, sollte man das Signal Adress-Strobe an Pin 2 des Testfeldes ansehen. Hier muß reges unregelmäßiges Auf und Ab zu sehen sein. Auch auf dem Datenbus herrscht reges Treiben, die Pinbelegung der CPU können Sie übrigens dem Schaltplan entnehmen.
Wenn Sie dennoch nicht weiterkommen, können Sie den Reparaturservice des Herstellers in Anspruch nehmen. Die Adresse lautet:

Firma
M.T. Himmeröder
Longbentonstr. 12
4353 Oer-Erkenschwick

Der Preis beträgt pauschal DM 89,- zuzüglich evtl. nötiger Bauteile.

5.6 Die Schnittstellen

5.6.1 Funktion der Schnittstellen

Die Programmierung der Schnittstellen-Bausteine ist mit allen Feinheiten nur anhand der Datenblätter möglich. Weil diese weit über 100 Seiten Umfang haben, können wir hier nur auf die elementaren Funktionen beispielhaft eingehen. Interessierten möchten wir deshalb die Anschaffung der Datenblätter (nicht ganz einfach) oder den Kauf eines Buches empfehlen.
Die Ports des 68230 lassen sich ohne große Vorbereitung zur Ausgabe von Binärmustern verwenden. Soll die C'T KAT-Ce z.B. einen Schrittmotor an Port A steuern, so muß nur das Datenrichtungsregister (DDRA) geladen werden. Ein kleines Beispielprogramm kann dies verdeutlichen. Es gibt an Port A abwechselnd $00 und $FF aus. Die Handshake-Leitungen bleiben unbenutzt. Die sich ergebende Rechteckschwingung kann an Pin 2 bis 9 von Port A mit dem Oszillographen beobachtet werden. Das Programm wird mit der Reset-Taste beendet.

```
DDRA    EQU     $400001
PORTA   EQU     $400011

        MOVE.B  #$FF,DDRA       ; Datenrichtung auf Ausgabe
LOOP    MOVE.B  #$00,PORTA
        MOVE.B  #$FF,PORTA
        BRA     LOOP
```

An der Frequenz der Schwingung läßt sich erkennen, wie schnell die C'T KAT-Ce ist. Die Schleife wird etwß 600.000 mal pro Sekunde durchlaufen.
Ein weiteres Beispiel zeigt die Datenausgabe über Port A unter Berücksichtigung der Handshake-Leitungen. Die Datenausgabe eines Bytes wird nun durch einen negativen Impuls an Pin 1 von Port A angezeigt und muß mit einem Low/High Übergang an Pin 10 bestätigt werden. Für unser Beispiel läßt sich dieses Handchake am besten durch Verbinden von Pin 1 und 10 simulieren. Ferner wird alle 100000 Schleifendurchläufe die Tastatur abgefragt. Das Programm kann dadurch per Tastendruck beendet werden.

```
KEYPRESS   EQU      $800110
PCR        EQU      $400001          ; allg. Port-Control-Register
PACR       EQU      $40000D          ; Port A Control-Register
DDRA       EQU      $400005          ; Datenrichtungs-Register
PORTA      EQU      $400011          ; Ausgabe-Port

           MOVE.B   #%00010000,PCR   ; unidirektional 8 Bit, H1 H2 enable
           MOVE.B   #%01111000,PACR  ; Submode 01, H2 pulsed H-Chake-Outp.
           MOVE.B   #$FF,DDRA
           MOVEQ    #$00,D1
           MOVEQ    #$FF,D2
LOOP1      MOVE.L   #100000,D3
LOOP2      MOVE.B   D1,PORTA
           MOVE.B   D2,PORTA
           SUBQ.L   #1,D3
           BNE      LOOP2
           JSR      KEYPRESS
           CMPI     #0,D0
           BEQ      LOOP1
           RTS
```

Das nächste Beispiel zeigt eine Anwendung des im 68230 enthaltenen Timers. Die 'Zeitmessung' in diesem Timer erfolgt durch Abwärtszählen des im Count-Registers enthaltenen Wertes. Erreicht der Zählerstand Null, wird ein Flag gesetzt und das Count-Register mit dem Inhalt des Preload-Registers nachgeladen. Count- und Preload-Register sind jeweils 3 Bytes lang. Der Zähltakt beträgt 1/32 des System-Taktes, in unserem Fall also 4 Mikrosek. Im folgenden Programm wird mit Hilfe des Timers eine in den Feldern STUNDE, MINUTE und SEKUNDE vorgegebene Zeit im Sekundentakt weitergezählt und auf dem Bildschirm ausgegeben. Drücken einer beliebigen Taste beendet das Programm.

```
TCR        EQU $400021 ; Timer Control Register

CPRH       EQU      $400027          ; Preload Register
CPRM       EQU      $400029
CPRL       EQU      $40002B

CNTRH      EQU      $40002F          ; Count Register
CNTRM      EQU      $400031
CNTRL      EQU      $400033

TSR        EQU      $400035          ; Timer Status Register
```

```
KEYPRESS  EQU     $800110
ASCIOUT   EQU     $800124
CLBILD    EQU     $80013C
GOTOXY    EQU     $800148

          BRA     PROGANF

STUNDE    DC.B    19              ; hier Startzeit
MINUTE    DC.B    36              ; vorgeben
SEKUNDE   DC.B    00
          SYNC

PROGANF   JSR     CLBILD          ; Schirm löschen

          MOVE.L  #250000,D0      ; 250000 X 4 = 1 Sek.
          LEA     CPRH-2,A0
          MOVEP.L D0,0(A0)        ; 250000 ♦ Preload Register
          MOVE.B  #%01000001,TCR  ; PC3 = TOUT, Timer enable

TIMEOUT   MOVE    #$0A10,D1
          JSR     GOTOXY          ; Cursor ♦ Spalte 10, Zeile 16

          MOVE.B  #'.',D2         ; Ausgabe der Zeit ♦ Schirm
          MOVE.B  STUNDE,D1
          JSR     DUALDEZ
          MOVE.B  MINUTE,D1
          JSR     DUALDEZ
          MOVE.B  SEKUNDE,D1
          JSR     DUALDEZ

          JSR     KEYPRESS
          CMPI.B  #0,D0
          BNE     ENDE

WAIT      MOVE.B  TSR,D0
          BEQ     WAIT            ; auf Flag warten
          MOVE.B  D0,TSR          ; Flag löschen

          ADDI.B  #1,SEKUNDE      ; Zeit weiterzählen
          CMPI.B  #60,SEKUNDE
          BNE     TIMEOUT
          CLR.B   SEKUNDE
          ADDI.B  #1,MINUTE
          CMPI.B  #60,MINUTE
          BNE     TIMEOUT
          CLR.B   MINUTE
          ADDI.B  #1,STUNDE
          CMPI.B  #24,STUNDE
```

```
                BNE         TIMEOUT
                CLR.B       STUNDE
BRA             TIMEOUT

DUALDEZ         ANDI.L      #$000000FF,D1       ; D1 DUAL ⧫ DEZ konvertieren
                DIVU        #10,D1
                MOVE.B      D1,D0
                ORI.B       #'0',D0             ; Nibble ⧫ ASCII
                JSR         ASCIOUT             ; 1.Ziffer ⧫ BS
                SWAP        D1
                MOVE        D1,D0
                ORI.B       #'0',D0
                JSR         ASCIOUT             ; 2.Ziffer ⧫ BS
                MOVE.B      D2,D0
                JSR         ASCIOUT             ; Punkt oder Blank ⧫ BS
ENDE            RTS
```

Bei der seriellen Datenübertragung hat sich ein Standard eingebürgert, der bei der Benutzung der Monitorroutinen problemlos genutzt werden kann. Aus diesem Grund beschränken wir uns darauf, bei den seriellen Schnittstellen die Bedeutung und die Besonderheiten der Signalleitungen zu erläutern.
An der 25 poligen V24-Buchse der C'T KAT-Ce sind Anschlüsse wie folgt belegt:

Pin2: TxD
Ausgabe der Daten. Es werden je Byte 1 Startbit, 8 Datenbits mit LSB vorweg, und 1 Stoppbit gesendet.

Pin 3: RxD
Eingang zum Empfang der Daten. Als Datenformat gilt 1 Startbit, 8 Datenbits mit LSB vorweg und 1 Stoppbit als vereinbart.

Pin4: RTS
High-Pegel an diesem Ausgang zeigt an, daß die C'T KAT-Ce bereit ist, Daten zu empfangen.

Pin5: CTS
An diesem Eingang signalisiert der Empfänger der Gegenseite durch High-Pegel, daß die C'T KAT-Ce Daten senden kann.

Pin7: GND

Pin8: DCD
An diesem Eingang meldet ein angeschlossenes Modem durch Anlegen von High-Pegel, daß das Trägersignal erkannt wurde. Falls nicht mit einem Modem gearbeitet wird, ist diese Leitung im Stecker mit Pin 20 zu verbinden.

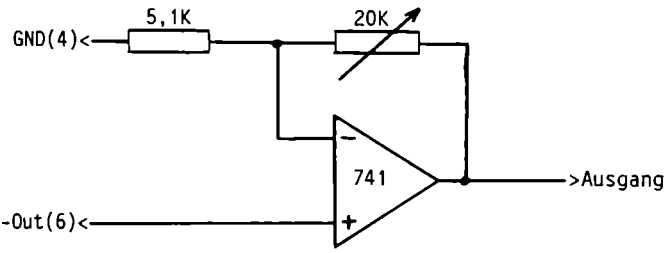
Pin 20: C'T KAT-Ce meldet sich durch High-Pegel bereit.

Für die Schnittstelle SERIELL2 gelten die oben gemachten Angaben entsprechend. Die Pinbelegung dieser Schnittstelle kann unter 5.6.4 nachgesehen werden.

Für die Bedienung des AD- und des DA-Wandlers sind im Monitor geeignete Routinen vorhanden. Hier noch einige Worte zur Äußeren Beschaltung dieser Wandler.
Der DA-Wandler liefert an seinem Ausgang für den Digitalwert 0 bis 255 eine Spannung

von 0 bis 2,55 Volt. Der Innenwiderstand des Ausgangs wird im Datenblatt mit 4K angegeben. Andere Ausgangsspannungen können mit einem externen Verstärker erzeugt werden. In dem hierzu gezeigten Schaltungsvorschlag kann mit dem Potentiometer die Verstärkung eingestellt werden. Die hochohmige Ankopplung an den DA-Wandler eliminiert dessen Innenwiderstand weitgehend.

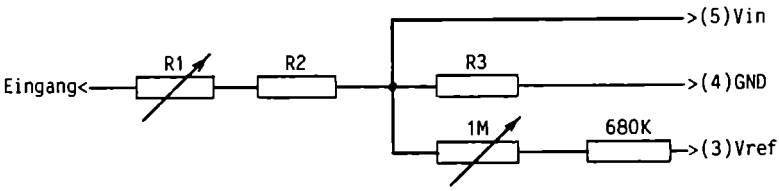
Ausgangsverstärker für DA-Wandler



Die Verstärkung ist 1:4 regelbar.

Der AD-Wandler ist für eine Eingangsspannung von 0 bis 2,55 Volt ausgelegt. Die Konvertierung in den entsprechenden Digitalwert 0 bis 255 dauert 14 Mikrosekunden. Der Eingangswiderstand beträgt 100K und die zulässige Eingangsspannung -0,5 bis +3,5 Volt.
Eine Änderung des Eingangsspannungs-Bereiches ist durch einen vorgeschalteten Spannungteiler möglich. Die Zeichnung gibt an, wie es geht.

Eingangsspannungsteiler für AD-Wandler

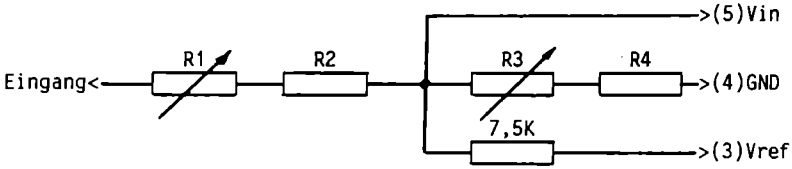


Eingangsspannungsbereich

+5 Volt	+10 Volt
R1 = 5K	R1 = 10K
R2 = 5,6K	R2 = 12K
R3 = 8,2K	R3 = 5,6K

Da die Eingangsspannung den Bereich -0,5 bis +3,5 Volt nicht überschreiten darf, ist es sinnvoll und genauer, einen Verstärker vorzuschalten.
Der Eingangsspannungs-Bereich kann auch nullsymetrisch aufgeteilt werden. Wie die Zeichnung zeigt, geschieht dies durch Anheben des Wandlereingangs. Daraus folgt, daß der gelieferte Digitalwert um $80 zu groß ist. Um den richtigen Wert mit Vorzeichen zu erhalten, muß man also $80 subtrahieren oder das MSB invertieren.

Eingangsspannungsteiler für bipolare Anwendung



Eingangsspannungsbereich

+/-5 Volt	+/-10 Volt
R1 = 5K	R1 = 10K Abgleich -Umax
R2 = 12K	R2 = 27K
R3 = 5K	R3 = 5K Abgleich +Umax
R4 = 12K	R4 = 8,2K

5.6.2 Interrupt-Steuerung

Zur Auslösung von Interrupts besitzt der MC68000 die Eingänge IPL0, IPL1 und IPL2. Die Signale an diesen Eingängen, Low-Pegel wird als 1 gewertet, werden vom Prozessor als Binärzahl interpretiert und es können somit Interrupt-Anforderungen von 7 verschiedenen Quellenbearbeitet werden. Dies setzt aber voraus, daß ein Prioritätsenkoder (74148) die 7 Leitungen zu 3 Bit codiert.
Im ROM ist diesen Anforderungen ein Vektor mit 7 Adressen zugeordnet. Kommt es zu einem Interrupt, so wählt der Prozessor anhand von IPL0 bis IPL2 eine Adresse aus und setzt das Programm an dieser Stelle fort.
Auf den Prioritätsenkoder wurde bei der C'T KAT-Ce verzichtet. Statt dessen sind über Jumper 3 Interruptleitungen direkt auf die IPL-Eingänge gelegt. Das Jumperfeld befindet auf der Platine parallel zu den CPU-Pins 27 bis 32.
Die Jumper sind bei Bedarf wie folgt zu stecken:

Postion	Verbindung	Wertigkeit
1	PC5/PIRQ(68230) ⧫ IPL2	4
2	keine Verbindung	
3	PC3/TOUT(68230) ⧫ IPL1	2
4	keine Verbindung	
5	INTRN(68681) ⧫ IPL0	1
6	keine Verbindung	

Unter der Voraussetzung, daß immer nur eine Leitung auf Low geht, werden aus dem Interrupt-Vektor nur die Adressen 1, 2 oder 4 ausgewertet. Die übrigen Adressen kommen nur dann zum Zuge, wenn 2 oder 3 Leitungen gleichzeitig ein Signal führten. In unserem Falle bedeutet das, daß 2 oder 3 Signale gleichzeitig eine Interrupt-Anforderung gestellt haben. Wenn dieser Fall auftritt, müssen wir entscheiden, in welcher Reihenfolge die Anforderungen bearbeitet werden. Damit geben wir den Interrupt-Quellen eine Priorität.
Ordnen wir den Bearbeitungs-Routinen für die drei InterruptQuellen INTRN(68681), TOUT(68230) und PIRQ(68230) die Namen A, B und C zu, so sind folgende Prioritäten möglich:

ABC ACB BAC BCA CAB CBA

Abhängig von der gewählten Priorität können Sie aus der nachfolgenden Tabelle ersehen, welche Bearbeitungsroutinen in welcher Reihenfolge bei welchem Interrupt ausgeführt werden müssen.

	IRQ	PRIORITÄT					
Wert	Vektor-Adr.	ABC	ACB	BAC	BCA	CAB	CBA
1	$64	A	A	A	A	A	A
2	$68	B	B	B	B	B	B
3	$6C	AB	AB	BA	BA	AB	BA
4	$70	C	C	C	C	C	C
5	$74	AC	AC	AC	CA	CA	CA
6	$78	BC	CB	BC	BC	CB	CB
7	$7C	ABC	ACB	BAC	BCA	CAB	CBA

Zur Verdeutlichung der Interruptbearbeitung wurde als Beispiel wieder ein kleines Uhrzeit-Programm gewählt, nur daß jetzt ein laufendes Programm jede Sekunde zum Weiterzählen und zur Ausgabe der Zeit unterbrochen wird. Sollten Sie ein ähnliches Problem programmieren wollen, so beachten Sie bitte, daß gleiche Monitor-Routinen nicht im Haupt- und Interruptprogramm gleichzeitig verwendet werden dürfen. Das hat nämlich zur Folge, daß die Routinen per Interrupt unterbrochen und dann während der Unterbrechung erneut angesprungen werden. Dß die Monitor-Routinen weder 'reenterable' noch gegen Unterbrechungen geschützt sind, muß diese Anwendung vermieden werden.
Das Zeitprogramm löscht zuerst den Bildschirm, stellt den Interruptzeiger für IPL1 und startet den Timer. Er ist jetzt so programmiert, daß die TOUT-Leitung (Timer Out) bei jedem Nulldurchgang des Zählers solange auf Low geht, bis das Flag im Timer-Status-Register vom Programm wieder gelöscht wird. Danach zählt das Programm in einer Schleife ein Register hoch und gibt den Wert über Port A aus. Zusätzlich wird jede Sekunde die Tastatur abgefragt. Ein Tastendruck beendet das Programm. Dß der Timer auch nach Beendigung des Programms weiterläuft, unterbindet das Programm vor seiner Beendigung weitere Interrupts durch Setzen der entsprechenden Bits im Statusregister der CPU. Mehr hierzu im Datenblatt.

```
DDRA    EQU     $400005
PORTA   EQU     $400011

TCR     EQU     $400021         ;Timer Control Register
```

```
CPRH      EQU     $400027          ; Preload Register
TSR       EQU     $400035          ; Timer Status Register

INTVEKT   EQU     $68

KEYPRESS  EQU     $800110
ASCIOUT   EQU     $800124
CLBILD    EQU     $80013C
GOTOXY    EQU     $800148

          BRA     PROGANF

STUNDE    DC.B    18               ; hier Startzeit
MINUTE    DC.B    25               ; vorgeben
SEKUNDE   DC.B    55

INTFLAG   DC.B    0
          SYNC

PROGANF   JSR     CLBILD           ; Bildschirm löschen
          MOVE.L  #INTERRUPT,INTVEKT ; Zeiger ♦ IRQ-Rout.
          MOVE.B  #$FF,DDRA        ; Port A auf Ausgabe

          MOVE.L  #250000,D0       ; Timer Starten
          LEA     CPRH-2,A0
          MOVEP.L D0,0(A0)
          MOVE.B  #%11100001,TCR

          ANDI.W  #%1111100011111  111,SR ; IRQ freigeben

          CLR.L   D1
ENDLOS    ADDI.B  #1,D1
          MOVE.B  D1,PORTA
          CMPI.B  #0,INTFLAG       ; IRQ gewesen ?
          BEQ     ENDLOS
          CLR.B   INTFLAG
          JSR     KEYPRESS
          CMPI.B  #0,D0
          BEQ     ENDLOS
          ORI.W   #%0000011100000  000,SR ; IRQ sperren
          RTS
; Beginn der Interrupt-Bearbeitung
INTERRUPT MOVEM.L A0-A6/D0-D7,-(SP) ; Register retten
          MOVE.B  #1,TSR           ; Timer Flag löschen
          ADDI.B  #1,SEKUNDE       ; Zeit weiterstellen
          CMPI.B  #60,SEKUNDE
          BNE     TIMEOUT
          CLR.B   SEKUNDE
```

```
            ADDI.B    #1,MINUTE
            CMPI.B    #60,MINUTE
            BNE       TIMEOUT
            CLR.B     MINUTE
            ADDI.B    #1,STUNDE
            CMPI.B    #24,STUNDE
            BNE       TIMEOUT
            CLR.B     STUNDE

TIMEOUT     MOVE      #$0A10,D1
            JSR       GOTOXY

            MOVE.B    #'.',D2
            MOVE.B    STUNDE,D1
            JSR       DUALDEZ
            MOVE.B    MINUTE,D1
            JSR       DUALDEZ
            MOVE.B    #':',D2
            MOVE.B    SEKUNDE,D1
            JSR       DUALDEZ
            MOVE.B    #$FF,INTFLAG      ; Flag für Hauptprogramm
            MOVEM.L   (SP)+,A0-A6/D0-D7
            RTE                         ; retour zum Hauptprogramm

DUALDEZ     ANDI.L    #$000000FF,D1
            DIVU      #10,D1
            MOVE.B    D1,D0
            ORI.B     #'0',D0
            JSR       ASCIOUT
            SWAP      D1
            MOVE.B    D1,D0
            ORI.B     #'0',D0
            JSR       ASCIOUT
            MOVE.B    D2,D0
            JSR       ASCIOUT
            RTS
```

Bitte beachten Sie, daß in Position 3 des Jumperfeldes ein Jumper gesteckt werden muß.

5.6.3 Register der Schnittstellen-Bausteine

68230 $400001: Port General Control Register
$400003: Port Service Request Register
$400005: Port A Data Direction Register
$400007: Port B Data Direction Register
$400009: Port C Data Direction Register

$40000B: Port Interrupt Vector Register
$40000D: Port A Control Register
$40000F: Port B Control Register
$400011: Port A Data Register
$400013: Port B Data Register
$400015: Port A Alternate Register
$400017: Port B Alternate Register
$400019: Port C Data Register
$40001B: Port Status Register
$40001D: ----------
$40001F: ----------
$400021: Timer Control Register
$400023: Timer Interrupt Vector Register
$400025: ----------
$400027: Counter Preload Register (High)
$400029: Counter Preload Register (Mid)
$40002B: Counter Preload Register (Low)
$40002D: ----------
$40002F: Count Register (High)
$400031: Count Register (Mid)
$400033: Count Register (Low)
$400035: Timer Status Register

68681

beim Lesen	beim Schreiben
$402001: Mode Reg. A (MR1A, MR2A)	Mode Reg. A (MR1A, MR2A)
$402003: Status Register A (SRA)	Clock Select Reg. A (CSRA)
$402005: Reserved	Command Register A (CRA)
$402007: RX Holding Reg. A (RHRA)	TX Holding Reg. A (THRA)
$402009: Inp.Port Change Reg.(IPCR)	Aux. Control Reg. (ACR)
$40200B: Interrupt Status Reg.(ISR)	Interrupt Mask Reg. (IMR)
$40200D: Counter/Timer Upper (CTU)	C/T Upper Register (CTUR)
$40200F: Counter/Timer Lower (CTL)	C/T Lower Register (CTLR)
$402011: Mode Reg. B (MR1B, MR2B)	Mode Reg. B (MR1B, MR2B)
$402013: Status Register B (SRB)	Clock Select Reg. B (CSRB)
$402015: Reserved	Command Register B (CRB)
$402017: RX Holding Reg. B (RHRB)	TX Holding Reg. B (THRB)
$402019: Interrupt Vector Reg.(IVR)	Interrupt Vector Reg.(IVR)
$40201B: Input Port	Output Port Conf.Reg.(OPCR)
$40201D: Start Counter Command	Set Output-Port Bits Command
$40201F: Stop Counter Command	Res. Outp.-Port Bits Command

5.6.4 Pinbelegung der Schnittstellen

68230 Port A — Printer-Anschluß

1	H2	;Strobe	11	GND
2	PA0	;D0	12	GND
3	PA1	;D1	13	GND
4	PA2	;D2	14	GND
5	PA3	;D3	15	GND
6	PA4	;D4	16	GND
7	PA5	;D5	17	GND
8	PA6	;D6	18	GND
9	PA7	;D7	19	GND
10	H1	;ACKNLG,inv.	20	NC

68230 Port B

1	+5 Volt	8	+5 Volt
2	PB1	9	PB0
3	PB3	10	PB2
4	PB5	11	PB4
5	PB7	12	PB6
6	H4	13	H3
7	GND	14	GND

68230 Port C

1	PC6	;PIACK,inv.	6	PC7	;TIACK,inv.
2	PC4	;DMAREQ,inv.	7	PC5	;PIRQ, inv.
3	PC2	;TIN	8	PC3	;TOUT
4	PC0		9	PC1	
5	GND		10	GND	

68681 ser. A

1	IP4	;ClockInp. R	6	OP2	;ClockOutp. T
2	IP3	;ClockInp. T	7	IP0	;CTS
3	GND		8	OP0	;RTS
4	NC		9	RxDA	;RxD
5	OP4	;DTR	10	TxDA	;TxD

68681 ser. B

1	NC		2	TxDB	;TxD
3	RxDB	;RxD	4	OP1	;RTS
5	IP1	;CTS	6	NC	
7	GND		8	IP5	;ClockInp. T
9	NC		10	NC	
11	NC		12	IP2	;ClockInp. R
13	NC		14	NC	
15	NC		16	NC	
17	NC		18	NC	
19	NC		20	OP5	;DTR
21	NC		22	NC	
23	NC		24	OP3	;ClockOutp. R/T

AD- und DA-Wandler

1 +5 Volt	6 DA-Outp.
2 VREF DA	7 -12 Volt
3 VREF AD	8 +12 Volt
4 GND	9 OP6 (68681) *)
5 AD-Inp.	10 AD-Busy,inv.

*) mit OP6(68681) kann event. ein externer Schalter bedient werden, z.B. Multiplexer.

Expansionsschnittstelle

1 D3	18 D4
2 D2	19 D5
3 D1	20 D6
4 D0	21 D7
5 A11	22 A1
6 Reset,inv.	23 A2
7 IPL1	24 A3
8 ExpSel,inv.	25 A4
9 R/W,inv.	26 A5
10A10	27 A6
11A9	28 A7
12E-Clock	29 A8
13A13	30 A12
14+5 Volt	31 GND
15+5 Volt	32 GND
16+5 Volt	33 GND
17+5 Volt	34 GND

Der Monitor der C'T KAT-Ce

Wie vieles im Leben wird der Monitor im Laufe der Zeit einige Wandlungen durchlaufen. Aus diesem Grund kann hier nicht die jeweils aktuelle Version abgedruckt werden. Sogar die beiden Eprom Ausführungen parallel bzw. seriell verändern die gesamten Adressen.
Es ist allerdings abzusehen, daß viele Konstanten unverändert bleiben und auch die Sprungtabelle in ihrer Form erhalten bleiben wird.
Damit steht fest, daß allenfalls weitere Konstanten und Sprünge hinzugefügt werden, was im einzelnen jeweils dokumentiert werden muß.
Weil nun bei jeder Änderung alle Adressen hinter der neu gefaßten Stelle neue Werte bekommen, ist es nicht sinnvoll, die Routinen incl. des erzeugten Codes abzudrucken. Wir haben daher, abgesehen von den Konstanten und der Sprungtabelle, die in übersetzter Form unten abgedruckt sind, nur das Textfile des zur Zeit der Drucklegung aktuellen Fassung unten wiedergegeben. Es ist davon auszugehen, daß die meisten Routinen auch zukünftig im wesentlichen bestehen bleiben, so daß mithilfe der Sprungtabelle auch bei neuer Adressenlage die gewünschte Stelle schnell aufgespürt werden kann.

Anhang

```
000000:             ;*********************  **********
000000:             ;*                       *
000000:             ;*                       KAT-Ce BOOTROM     *
000000:             ;*                       V#000008           *
000000:             ;*                       23.07.86           *
000000:             ;*                       *
000000:             ;*********************  **********
000000:             ;
000000:             Parallel        EQU 1
000000:             ;
000000:             RAMSTART        EQU $000000
000000:             STARTADR        EQU $800000
000000:             ;
000000:00000A00     STACKPTR        EQU RAMSTART+$000A00
000000:00020000     RAMMAX          EQU RAMSTART+$020000
000000:00003FFC     zwei6264        EQU RAMSTART+$3FFC
000000:00007FFC     vier6264        EQU RAMSTART+$7FFC
000000:0000FFFC     zwei43256       EQU RAMSTART+$FFFC
000000:0001FFFC     vier43256       EQU RAMSTART+$1FFFC
000000:00018000     CodeStMax       EQU RAMSTART+$18000
000000:             ;
000000:12345678     RESETB          EQU $12345678           ;EOR MIT RESETADR
000000:             ;
000000:             ;
```

```
000000:                 ; RAM Adressvektoren
000000:                 ;
000000:00000000 Exceptions      EQU RAMSTART            ;niedrigste RAM-Adresse
000000:00000100 DISKB           EQU exceptions+$100
000000:00000300 serbBaud        EQU DiskB+$200          ;merkt sich die Baudrate vor Reset
000000:00000301 seraBaud        EQU serbBaud+1
000000:00000302 FileFlags       EQU seraBaud+1          ;setzt Bits fuer geoeffnete Files
000000:00000304 ymax            equ  FileFlags+2        ;max x-Koord.
000000:00000305 xmax            equ  ymax+1             ;max y-Koord.
000000:00000306 StartA7         EQU xmax+1
000000:0000030A BUFFER          EQU StartA7+4           ;fuer Eingabe einer Textzeile
000000:0000035A REGSAV          EQU BUFFER+80           ;Speicherplatz fuer 68000 Register
000000:0000039A STATSAV         EQU REGSAV+$40
000000:0000039C REGFLAG         EQU STATSAV+2
000000:000003A0 MONFLAGS        EQU REGFLAG+4
000000:000003A2 SAVER           EQU MONFLAGS+2
000000:000003A4 ZNUMMER         EQU SAVER+2
000000:000003A8 OUTVECT         EQU ZNUMMER+4
000000:000003AC YKOORD          EQU OUTVECT+4
000000:000003AD XKOORD          EQU YKOORD+1
000000:000003AE BILDGR          EQU XKOORD+1
000000:000003B0 OFFSET          EQU BILDGR+2
000000:000003B4 TEXTSTART       EQU OFFSET+4
000000:000003B8 RAMENDE         EQU TEXTSTART+4
000000:000003BC CODESTART       EQU RAMENDE+4
000000:000003C0 A0SPEICH        EQU Codestart+4
000000:000003C4 RESETCMP        EQU A0SPEICH+4
000000:000003C8 RESETADR        EQU RESETCMP+4
000000:
000000:000003CC NachMonVar      EQU ResetAdr+4
000000:                 ;
000000:                 ;Adressen mit 'RIEN' sind vorlaeufig nicht definiert
000000:                 ;und verzweigen zur Begruessungsmeldung der KAT-Ce
000000:                 ;
000000:                 ORG STARTADR
800000:                 ; MC 68000 Vector Tabelle
800000:00000A00 DC.L STACKPTR
800000:008001D4 DC.L PREINIT
800000:008008B4 DC.L Rien               ;Busfehler nicht existent
800000:00800D22 DC.L ADRERR
800000:00800D2A DC.L ILLINSTR
800000:008008B4 DC.L rien               ;ZERDIV
800000:008008B4 DC.L rien               ;CHK
800000:008001CC DC.L TRAP               ;TRAPV
800000:008008B4 DC.L rien               ;Privilegierte Verletzung
800000:00800A5C DC.L TRACEit            ;Trace
800000:00800D2A DC.L ILLINSTR           ;EM1010
800000:00800D2A DC.L ILLINSTR           ;EM1111
800000:008008B4 DC.L Rien
800000:008008B4 DC.L Rien
800000:008008B4 DC.L Rien
800000:008008B4 DC.L Rien
```

```
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008001CE  DC.L  INTERRUPT   ;unechter Interrupt
800000:008001CE  DC.L  INTERRUPT   ;Level 1
800000:008001CE  DC.L  INTERRUPT   ;Level 2
800000:008001CE  DC.L  INTERRUPT   ;Level 3
800000:008001CE  DC.L  INTERRUPT   ;Level 4
800000:008001CE  DC.L  INTERRUPT   ;Level 5
800000:008001CE  DC.L  INTERRUPT   ;Level 6
800000:008001CE  DC.L  INTERRUPT   ;Level 7
800000:00800A3A  DC.L TRACEAUS     ;TRAP #0
800000:00800A4C  DC.L  TRACEEIN    ;TRAP #1
800000:008001CC  DC.L TRAP         ;TRAP #2
800000:008001CC  DC.L TRAP         ;TRAP #3
800000:008001CC  DC.L TRAP         ;TRAP #4
800000:008001CC  DC.L TRAP         ;TRAP #5
800000:008001CC  DC.L TRAP         ;TRAP #6
800000:008001CC  DC.L TRAP         ;TRAP #7
800000:008001CC  DC.L TRAP         ;TRAP #8
800000:008001CC  DC.L TRAP         ;TRAP #9
800000:008001CC  DC.L TRAP         ;TRAP #10
800000:008001CC  DC.L TRAP         ;TRAP #11
800000:008001CC  DC.L TRAP         ;TRAP #12
800000:008001CC  DC.L TRAP         ;TRAP #13
800000:008001CC  DC.L TRAP         ;TRAP #14
800000:008001CC  DC.L TRAP         ;TRAP #15
800000:
800000:          ;fuer spaetere Anwendungen:
800000:
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rien
800000:008008B4  DC.L Rin
800000:008008B4  DC.L Rien
800000:          ;
800000:          ;fuer WMG-Pas
800000:          ;
800000:00000468  DC.L filename
800000:000003F0  DC.L endezeiger
```

```
800000:00804210  DC.L warmedi
800000:008040F8  DC.L pasfehler
800000:          ;
800000:          ;Start der Sprungeve ktoren ab $800100
800000:          ;
800100:60000236  BRA.L SetResetAdr          ;ResetAdresse in A0
800104:600000CE  BRA.L PREINIT              ;Kaltstart
800108:60000780  BRA.L MONinit              ;Monitorroutine
80010C:6000060C  BRA.L QUIT
800110:600002B0  BRA.L KEYPRE SS            ;Anzahl der anstehenden Bytes in D0
800114:600001B0  BRA.L BYTEIN               ;Byte in D0
800118:600001B4  BRA.L TASTIN               ;liest Zeichen von Tastatur
80011C:600001B6  BRA.L  GETSTRING           ;A1:Zeiger auf Anfang, 1. Byte Laenge
800120:600001A8  BRA.L BYTEOUT              ;Byte in D0
800124:60000236  BRA.L  ASCIOUT             ;Ausgabezeichen in D0
800128:60000224  BRA.L WORDO UT             ;Wort in D0, D0 ist veraendert
80012C:6000032C  BRA.L PRTBYT               ;Byte in D1 als Hexbyte ausgeben, D0 veraendert
800130:60000320  BRA.L  PRTWORD             ;Wort in D1 Hex ausgeben, D0 v.
800134:60000EC0  BRA.L STROUT               ;A1 Zeiger auf String, im 1 Byte
800134:                                     ; muss die Laenge stehen
800138:600002F4  BRA.L CROUT                ;Carr. Return, Linefeed, D0 v.
80013C:600002FC  BRA.L CLBILD               ;Bildschirm loeschen, D0 v.
800140:600002E8  BRA.L PRTCOL               ;':' schreiben, D0 v.
800144:600002FA  BRA.L PRTMSP               ;(D1)+1 Leerzeichen, D0-D1 v.
800148:60000DA6  BRA.L GOTOX  Y             ;D1.W enthaelt in Lowbyte x-K
800148:                                     ;im Highbyte y-K., D0-D2 v.
80014C:600002D4  BRA.L BELL
800150:600011DE  BRA.L DISZEIL  E           ;Adresse in A1, A1 zeigt anschliessend
800150:                                     ;auf neue Adresse, A0-A4, D0-D4 veraendert
800154:60000E08  BRA.L RESETF               ;Reset File, A1 Zeiger auf String
800154:                                     ;wie bei STROUT, Filenr in D1, D0-D2, A1 v.
800158:60000D88  BRA.L  REWRITE             ;wie bei RESETF
80015C:60000E3A  BRA.L BLOCKR               ;Filenummer in D1, D0-D2 v.
800160:60000E14  BRA.L  BLOCKW              ;wie bei BLOCKR
800164:60000DE8  BRA.L CLOSE                ;
800168:60000DEC  BRA.L Closedel             ;schliesst und loescht File
80016C:60000DD2  BRA.L  IORESULT            ;liest IORes der letzten Diskettenoper. in D0
800170:60000E48  BRA.L  UNITOUT             ;Ausgabebyte in D0
800174:60000E54  BRA.L UNITINIT             ;initialisiert Hostcomputer-Drucker
800178:600010C4  BRA.L DAOut                ;Ausgabewert in D0
80017C:600010AA  BRA.L ADIn                 ;Wert in D0
800180:60001044  BRA.L PARAINI T            ;INIT PARALLEL A
800184:60001090  BRA.L  PARASTAT            ;Bit 0 von D0=1: Byte kann gesendet werden
800188:6000105E  BRA.L  PARAOUT             ;Byte in D0
80018C:60001094  BRA.L PARAIN
800190:60001032  BRA.L  PARBINIT            ;INIT PARALLEL B
800194:60001090  BRA.L  PARBSTAT
800198:6000104C  BRA.L PARBO  UT            ;Byte in D0
80019C:60001086  BRA.L PARBIN
8001A0:60000F4E  BRA.L  SERAINIT            ;INIT Seriellport A, D1: Baudrate: $CC,$44
8001A4:60000FF2  BRA.L  SERASTAT            ;Bit 0 von D0=1: Byte kann gesendet werden
8001A4:                                     ;Bit 1 von D0=1: Byte empfangen
```

```
8001A8:60000FAA  BRA.L SERAOUT                    ;BYTE IN D0
8001AC:60000FC8 BRA.L SERAIN                      ;BYTE IN D0
8001B0:60000F4E  BRA.L SERBINIT                   ;
8001B4:60001004  BRA.L SERBSTAT
8001B8:60000FA4  BRA.L  SERBOUT
8001BC:60000FC2 BRA.L SERBIN
8001C0:60001084  BRA.L initportb
8001C4:600010A2  BRA.L readportb
8001C8:600010F8  BRA.L sendportb



nolist
;*********************************
;*
;*KAT-Ce BOOTROM
;* V#000008
;* 23.07.86
;*
;*********************************
;
;
Parallel                EQU     1
;
RAMSTART                EQU     $000000
STARTADR                EQU     $800000
;
;
STACKPTR                EQU     RAMSTART+$000A00
RAMMAX                  EQU     RAMSTART+$020000
zwei6264                EQU     RAMSTART+$3FFC
vier6264                EQU     RAMSTART+$7FFC
zwei43256               EQU     RAMSTART+$FFFC
vier43256               EQU     RAMSTART+$1FFFC
CodeStMax               EQU     RAMSTART+$18000
;
RESETB                  EQU     $12345678       ;EOR MIT RESETADR
;
;
;
;
; RAM Adressvektoren
;
Exceptions              EQU     RAMSTART        ;niedrigste RAM-Adresse
DISKB                   EQU     exceptions+$100
serbBaud                EQU     DiskB+$200      ;merkt sich die Baudrate vor Reset
seraBaud                EQU     serbBaud+1
FileFlags               EQU     seraBaud+1      ;setzt Bits fuer geoeffnete Files
ymax                    equ     FileFlags+2     ;max x-Koord.
xmax                    equ     ymax+1          ;max y-Koord.
StartA7                 EQU     xmax+1
```

```
BUFFER          EQU   StartA7+4          ;fuer Eingabe einer Textzeile
REGSAV          EQU   BUFFER+80          ;Speicherplatz fuer 68000 Register
STATSAV         EQU   REGSAV+$40
REGFLAG         EQU   STATSAV+2
MONFLAGS        EQU   REGFLAG+4
SAVER           EQU   MONFLAGS+2
ZNUMMER         EQU   SAVER+2
OUTVECT         EQU   ZNUMMER+4
YKOORD          EQU   OUTVECT+4
XKOORD          EQU   YKOORD+1
BILDGR          EQU   XKOORD+1

OFFSET          EQU   BILDGR+2
TEXTSTART       EQU   OFFSET+4
RAMENDE         EQU   TEXTSTART+4
CODESTART       EQU   RAMENDE+4
A0SPEICH        EQU   Codestart+4
RESETCMP        EQU   A0SPEICH+4
RESETADR        EQU   RESETCMP+4

NachMonVar      EQU   ResetAdr+4
;
;Adressen mit 'RIEN' sind vorlaeufig nicht definiert
;und verzweigen zur Begruessungsmeldung der KAT-Ce
;
                ORG   STARTADR
; MC 68000 Vector Tabelle
                DC.L  STACKPTR
                DC.L  PREINIT
                DC.L  Rien             ;Busfehler nicht existent
                DC.L  ADRERR
                DC.L  ILLINSTR
                DC.L  rien             ;ZERDIV
                DC.L  rien             ;CHK
                DC.L  TRAP             ;TRAPV
                DC.L  rien             ;Privilegierte Verletzung
                DC.L  TRACEit          ;Trace
                DC.L  ILLINSTR         ;EM1010
                DC.L  ILLINSTR         ;EM1111
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  Rien
                DC.L  INTERRUPT        ;unechter Interrupt
```

```
DC.L   INTERRUPT          ;Level 1
DC.L   INTERRUPT          ;Level 2
DC.L   INTERRUPT          ;Level 3
DC.L   INTERRUPT          ;Level 4
DC.L   INTERRUPT          ;Level 5
DC.L   INTERRUPT          ;Level 6
DC.L   INTERRUPT          ;Level 7
DC.L   TRACEAUS           ;TRAP #0
DC.L   TRACEEIN           ;TRAP #1
DC.L   TRAP               ;TRAP #2
DC.L   TRAP               ;TRAP #3
DC.L   TRAP               ;TRAP #4
DC.L   TRAP               ;TRAP #5
DC.L   TRAP               ;TRAP #6
DC.L   TRAP               ;TRAP #7
DC.L   TRAP               ;TRAP #8
DC.L   TRAP               ;TRAP #9
DC.L   TRAP               ;TRAP #10
DC.L   TRAP               ;TRAP #11
DC.L   TRAP               ;TRAP #12
DC.L   TRAP               ;TRAP #13
DC.L   TRAP               ;TRAP #14
DC.L   TRAP               ;TRAP #15

;fuer spaetere Anwendungen:

DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
DC.L   Rien
;
;fuer WMG-Pas
;
DC.L   filename
DC.L   endezeiger
DC.L   warmedi
DC.L   turbofehler
;
;Start der Sprungevektoren ab #800100
;
bra.l  SetResetAdr        ;ResetAdresse in A0
BRA.L  PREINIT            ;Kaltstart
BRA.L  MONinit            ;Monitorroutine
BRA.L  QUIT
```

```
BRA.L  KEYPRESS     ;Anzahl der anstehenden Bytes in D0
BRA.L  BYTEIN       ;Byte in D0
BRA.L  TASTIN       ;liest Zeichen von Tastatur
BRA.L  GETSTRING    ;A1:Zeiger auf Anfang, 1. Byte Laenge
BRA.L  BYTEOUT      ;Byte in D0
BRA.L  ASCIOUT      ;Ausgabezeichen in D0
BRA.L  WORDOUT      ;Wort in D0, D0 ist veraendert
BRA.L  PRTBYT       ;Byte in D1 als Hexbyte ausgeben, D0 veraendert
BRA.L  PRTWORD      ;Wort in D1 Hex ausgeben, D0 v.
BRA.L  STROUT       ;A1 Zeiger auf String, im 1 Byte
                    ; muss die Laenge stehen
BRA.L  CROUT        ;Carr. Return, Linefeed, D0 v.
BRA.L  CLBILD       ;Bildschirm loeschen, D0 v.
BRA.L  PRTCOL       ;':' schreiben, D0 v.
BRA.L  PRTMSP       ;(D1)+1 Leerzeichen, D0-D1 v.
BRA.L  GOTOXY       ;D1.W enthaelt in Lowbyte x-K
                    ;im Highbyte y-K., D0-D2 v.
BRA.L  BELL
BRA.L  DISZEILE     ;Adresse in A1, A1 zeigt anschliessend
                    ;auf neue Adresse, A0-A4,D0-D4 veraendert
BRA.L  RESETF       ;Reset File, A1 Zeiger auf String
                    ;wie bei STROUT, Filenr in D1, D0-D2, A1 v.
BRA.L  REWRITE      ;wie bei RESETF
BRA.L  BLOCKR       ;Filenummer in D1, D0-D2 v.
BRA.L  BLOCKW       ;wie bei BLOCKR
BRA.L  CLOSE        ;
BRA.L  Closedel     ;schliesst und loescht File
BRA.L  IORESULT     ;liest IORes der letzten Diskettenoper. in D0
BRA.L  UNITOUT      ;Ausgabebyte in D0
BRA.L  UNITINIT     ;initialisiert Hostcomputer-Drucker
BRA.L  DAOut        ;Ausgabewert in D0
BRA.L  ADIn         ;Wert in D0
BRA.L  PARAINIT     ;INIT PARALLEL A
BRA.L  PARASTAT     ;Bit 0 von D0=1: Byte kann gesendet werden
BRA.L  PARAOUT      ;Byte in D0
BRA.L  PARAIN
BRA.L  PARBINIT     ;INIT PARALLEL B
BRA.L  PARBSTAT
BRA.L  PARBOUT      ;Byte in D0
BRA.L  PARBIN
BRA.L  SERAINIT     ;INIT Seriellport A, D1: Baudrate: $CC,$44
BRA.L  SERASTAT     ;Bit 0 von D0=1: Byte kann gesendet werden
                    ;Bit 1 von D0=1: Byte empfangen
BRA.L  SERAOUT      ;BYTE IN D0
BRA.L  SERAIN       ;BYTE IN D0
BRA.L  SERBINIT     ;
BRA.L  SERBSTAT
BRA.L  SERBOUT
BRA.L  SERBIN

bra.l  initportb
bra.l  readportb
```

```
                    bra.l   sendportb

                    nolist

TRAP                RTE                         ;nicht definiert
INTERRUPT           ori     #$7000,SR           ;Interrupt verhindern
                    RTE
;
; RAM Vector Tabelle initalisieren
;---------------------------------
PREINIT             MOVEM.L D0-A7,REGSAV

          if startadr-$800000
                    bra kaltst
          endif

                    MOVE.L RESETCMP,D0          ;PROBE AUF WARMSTART
                    EORI.L #RESETB,D0
                    CMP.L RESETADR,D0
                    BNE KALTST
                    MOVE.L D0,-(A7)             ;ResetAdresse auf Stack
          if parallel
                    bsr.linitportb
          else
                    move.B serbBaud,d1
                    bsr.l serbinit
          endif

                    RTS                         ;ResetRoutine starten

KALTST              ORI #$2700,SR               ;SUPERVISOR SETZEN, INTERRUPT=7
                    move.w #$BB44,SerBBaud      ;Voreinstellung SerA: 300 Baud
                                                ;          SerB: 9600 Baud
                    MOVE.L #RETURN,OUTVECT      ;RTS - BEFEHL
                    CLR FileFlags               ;alle Files geschlossen
                    MOVE.W #$0F05,BILDGR        ;80ZEICHEN
                    move.w #$1727,ymax          ;xmax,ymax vorsorglich setzen
                                                ;40/24
                    moveq #$40,D1 ;             $800000..$8000FF nach $0.. kopieren
                    lea StartAdr,A0
                    lea RamStart,A1
KopVektoren         move.l  (A0)+,(A1)+
                    subq    #1,d1
                    bne     KopVektoren

                    move.l  #STACKPTR,Textstart
;
;Ram-Anfang suchen
;-----------------

                    move.l  #CodeStMax,codestart;hoechstmoeglicher Wert
                    move.l  #vier43256,A0       ;hoechste Long-RamAdresse
```

```
                move.l  (a0),d0             ;Inhalt der Speicherstelle retten
                move.l  #ResetB,D1          ;EOR-Wert
                EOR.L   D1,D0               ;Alten Inhalt veraendern
                move.l  d0,(a0)             ;und neu speichern
                cmp.l   (a0),d0             ;liess sich dieser Wert setzen?
                bne     nur2Rams            ;nur 2 RAM's eingesetzt
                move.l  #vier6264,A0        ;Probe, ob 4 RAMs 6264
                cmp.l   (a0),D0             ;steht dort derselbe Wert?
                beq     d0zurueck           ;ja, Sprung, 4 RAM's 6264
                move.l  #vier43256,A0       ;sonst 4 RAMs 43256
                bra     d0zurueck           ;Sprung
nur2Rams        move.l  #zwei43256,A0       ;Probe, ob 6264 RAMs
                move.l  (a0),d0             ;hier kommt nur bei 43256 Wert
                eor.l   d1,d0               ;wiederum aendern
                move.l  d0,(a0)             ;setzen
                cmp.l   (a0),d0             ;identisch?
                beq     Foundzwei43256      ;ja, 2 RAMs 43256
                move.l  #zwei6264,A0        ;Ende fuer 2 6264 RAMs
                bra     RamEndsetzen        ;Sprung
Foundzwei43256  eor.l   d1,d0               ;alten Wert wiederherstellen
                move.l  d0,(a0)             ;und speichern
                move.l  #zwei43256,A0       ;wenn 2 43256 exist.
                bra     RamEndsetzen        ;RAMEnde setzen
d0zurueck       eor.l   d1,d0               ;alten Wert wiederherstellen
RamEndsetzen    move.l  d0,(a0)             ;und speichern
                addq.l  #4,A0
                move.l  a0,RamEnde          ;gefundene Adresse speichern
                cmpa.l  #zwei6264+4,A0      ;nur 2 RAMs 6264?
                bne     no2R6264            ;nein
                move.l  #zwei6264-$000FFC,codestart;setzen
no2R6264        cmpa.l  #vier6264+4,A0      ;4 RAMs 6264?
                bne     no4R6264            ;nein, dann gilt urspruengl. Wert
                move.l  #vier6264-$1FFC,codestart;bei 2 RAMs 6264
no4R6264        cmpa.l  #zwei43256+4,A0     ;nur 2 43256
                bne     RSfertig
                move.l  #zwei43256-$3FFC,codestart
RSfertig
                if startadr-$800000
                move.l  #vier6264-$7FC,codestart
                move.l  #endecode,textstart
                endif


;
;
; ASCII Konstanten und besondere Konstanten
;-----------------------------------------
JUMP            EQU     $4EF9               ;68000 Codefuer JMP.L
ESCE            EQU     29                  ;Loesche bis Zeilenende
BKSPC           EQU     $08                 ;Backspace
CR              EQU     $0D                 ;CARRIAGE RETURN
DC1             EQU     $11                 ;DC1 STEUERKOMMANDO
DC2             EQU     $12                 ;DC2 Steuerkommando
```

```
TRUNC           EQU   $11                ;Zeileneingabe abbrechen
RAROW           EQU   $15                ;CURSOR ↑
SPC             EQU   $20                ;Leerzeichen
;
;
                IF    Parallel
                bsr.L InitPortB
                bSR   BYTEIN
                BRA.L Init               ;Ueberspringe Ein/Ausgabe-Definition
Bytein          BRA.L ReadPortB
Byteout         BRA.L SendPortB

                ELSE

suchebdneu      MOVE.B#$DD,D1            ;Start mit 38400 Bd
suchebaud       move.B D1,serbBaud       ;Baudrate fuer RESET Warmstart merken
                bsr.L  SerBInit
                bsr.L  serbin            ;Byte lesen
                CMPI.B#$D,D0             ;ist es CR?
                BEQ    init              ;gefunden
                move   #50002,d2         ;Pause erzeugen
baudsuchpause   dbf    d2,baudsuchpause
                subi.b #$11,D1           ;naechste Baudrate einstellen
                CMPI.B#$33,D1            ;unter 300 Bd geht es nicht
                BNE    suchebaud         ;weitersuchen
                BEQ    suchebdneu        ;neu suchen

Byteout         BRA.L  SerBout

Bytein          BRA.L  SerBin
                ENDIF

tastin          moveq  #$13,D0           ;Anforderung: Taste
                bsr    byteout
                bra    bytein            ;Taste lesen

GetString       movem.l a0/a1/d0,-(a7)
                move.l a1,a0
                clr.b (a0)+
Getstrw         bsr tastin
                cmpi.b #$20,d0
                bcs GetScontrol
                move.b d0,(a0)+
                addq.b #1,(a1)
                bsr AsciOut
                bra Getstrw
GetScontrol     cmpi.b #$d,d0
                beq GetScr
                cmpi.b #$8,d0
                beq GetSlinks
GetSfehler      moveq #$7,d0
                bsr AsciOut
```

```
                bra Getstrw
GetSlinks       tst.b (a1)
                beq GetSfehler
                subq.b #1,(a1)
                subq.l #1,a0
                bsr AsciOut
                moveq #$20,d0
                bsr AsciOut
                moveq #$8,d0
                bsr AsciOut
                bra Getstrw
GetScr          move.b d0,(a0)
                bsr AsciOut
                moveq #$a,d0
                bsr AsciOut
                movem.l (a7)+,a0/a1/d0
                rts

INIT            MOVE.L #MONINIT,D0
                move.l  D0,ResetAdr
                eori.l  #ResetB,D0
                move.l  D0,ResetCmp
                bra.L   MONINIT
SetResetAdr     Move.l  A0,ResetAdr
                beq     NoResetAdr
                EXG.L   A0,D0
                Eori.l  #ResetB,D0
                EXG.L   D0,A0
NoResetAdr      Move.l  A0,ResetCmp
                RTS

;
; Sendet D0.W an Host
;--------------------
WORDOUT         MOVE.W D0,-(A7)         ;D0 auf dem Stack sichern
                LSR.W #8,D0             ;oberes Byte in unteres schieben
                BSR BYTEOUT             ;High-Byte ausgeben
                MOVE.W (A7)+,D0         ;D0 vom Stack holen
                BRA BYTEOUT             ;Low-Byte ausgeben

;
;
; SENDET D0.B AN HOST
;--------------------
ASCIOUT         MOVEM.L D1/A0,-(A7)
                CMP.B #$20,D0           ;Controllzeichen?
                BCC ASNORM              ;Buchstaben ausgeben
                CMP.B #$8,D0            ;BACKSPACE?
                BEQ ASBACK
                CMP.B #$D,D0            ;CR?
                BEQ ASCR
                CMP.B #$A,D0            ;LINEFEED?
                BEQ ASLF
```

```
                    CMP.B #$C,D0            ;BILDLOESCHEN?
                    BNE ASAUS
                    CLR.B YKOORD            ;Cursorposition in oberste Zeile
ASCR                CLR.B XKOORD            ;Cursorposition auf Zeilenanfang
                    BRA ASAUS
ASNORM              ADDQ.B #1,XKOORD        ;ERHOEHEN
                    move.b xkoord,d1
                    cmp.b xmax,d1           ;rechter Rand erreicht?
                    bgt APrOut              ;eventuell Druckerausgabe
                    bra asaus               ;Bildschirmausgabe

ASBACK              TST.B XKOORD            ;Cursor am Zeilenanfang?
                    BEQ ASAUS               ;weiter links wird's gefaehrlich
                    SUBQ.B #1,XKOORD        ;Cursor einen nach links
                    BRA ASAUS
ASLF                move.b ymax,d1
                    CMP.B YKOORD,D1         ;unteres Bildschirmende erreicht?
                    BEQ ASAUS               ;Ja
                    ADDQ.B #1,YKOORD        ;Zeile tiefer
ASAUS               BSR ByteOut             ;Zeichen auf dem Bildschirm ausgeben
APrOut              MOVE.L OUTVECT,A0       ;event. Druckerausgabe?
                    JSR (A0)                ;event. Zeichen drucken
                    MOVEM.L (A7)+,D1/A0     ;D1,A0 zurueckholen
RETURN              RTS

KEYPRESS            MOVEQ #$12,D0           ;Kommando fuer Keypress an Host
                    BSR BYTEOUT
                    BRA BYTEIN              ;Anzahl der vorhandenen Zeichen
;
;
; Tabelle der Monitor-Kommandos
;-----------------------------
CMDLST              DC.B 'M',':' ;MOVE      ;STORE BYTES
                    DC.B 'G','O'            ;GO ;OUTPUT CONTROL
                    DC.B 'Q','R'            ;QUIT ;REGISTER SET/DUMP
                    DC.B 'T','V'            ;TRACE ;VERIFY
                    DC.B 'X',$D             ;EXAMINE ;EXAMINE1
                    DC.B '?','Z'            ;RECHNEN ;DISASSEMBLER
                    DC.B 'P'                ;AUSGABEGERAETE
                    DC.B 'L','S'            ;LOAD ;Save
CMDEND              ;
                    sync
;
; Kommando Sprungtabelle
;----------------------
JMPLST              DC      MOVE
                    DC      SETMEM
                    DC      GO
                    DC      OUTPUT
                    DC      BEGRUESSUNG
                    DC      REGDSP
                    DC      TRACE
```

```
                DC      VERIFY
                DC      XAMINE
                DC      XAMINE1
                DC      RECHNEN
                DC      DISASS
                DC      AUSGABE
                DC      LESEN
                DC      SCHREIBE
;
; Tabelle zur Anzeige des Statusregisters (Name,Bitposition,0,Maske)
;----------------------------------------
SRDAT           DC.B    'T',15,0,1          ;Trace-Modus
                DC.B    'S',13,0,1          ;Supervisor-Modus
                DC.B    'I'                 ;Interrupt-Ebene
                DC.B    'X',4,0,1           ;Extend-Flag
                DC.B    'N',3,0,1           ;Negativ-Flag
                DC.B    'Z',2,0,1           ;Zero-Flag
                DC.B    'V',1,0,1           ;Overflow-Flag
                DC.B    'C',0,0,1           ;Carry-Flag
                sync
;
;
; Fehler durch Piep anzeigen und Ruecksprung zum Monitor
;-----------------------------------------------------
SUBERR          ADDQ.L #4,A7                ;Adresse der fehlerhaften Prozedur loeschen
ERROR           BRA     BELL                ;und piep sagen
;
; Sonderzeichen fuer den Bildschirm
;---------------------------------
cleoline        MOVEQ #ESCE,D0              ;bis Zeilenende loeschen
                BRA ASCIOUT1
BELL            MOVEQ #$7,D0                ;einen Piep ertoenen lassen
                BRA ASCIOUT1
PRTSPC          MOVEQ #SPC,D0               ;Leerzeichen ausgeben
                BRA ASCIOUT1
PRTCOL          MOVEQ #':',D0               ;: ausgeben
                BRA ASCIOUT1
CROUT           MOVEQ #CR,D0                ;Wagenruecklauf (?) und
                BSR ASCIOUT1
                MOVEQ #$A,D0                ;Zeilenvorschub ausgeben
                BRA ASCIOUT1
crout2          bsr crout                   ;Zwei Zeilenvorschuebe
                bra crout
CLBILD          MOVEQ #12,D0                ;Bildschirm loeschen
ASCIOUT1        BRA ASCIOUT
;
; erzeuge (D1)+1 Leerzeichen
;---------------------------
PRTMSP          MOVEQ #SPC,D0               ;Leerzeichen
PRTMSP1         BSR ASCIOUT1                ;ausgeben
                DBF D1,PRTMSP               ;bis D1 leergedbft ist
                RTS
```

```
;
; D2 LONG in hexadezimal ausgeben
;---------------------------------
PRTLONG         MOVE.L D2,D1            ;Kopie in D1 anlegen
                SWAP D1                 ;oberes Wort in unteres kopieren
                BSR PRTWORD             ;High-Wort ausgeben
                MOVE D2,D1              ;Low-Wort wiederholen und ausgeben
;
; Wort in D1 hexadezimal ausgeben
;---------------------------------
PRTWORD         MOVE.W D1,-(A7)         ;Wort auf den Stack sichern
                LSR #8,D1               ;High-Byte ins Low-Byte schieben
                BSR PRTBYT              ;und ausgeben
                MOVE.W (A7)+,D1         ;Wort vom Stack holen und Low-Byte
;
; Byte in D1 hexadezimal ausgeben
;---------------------------------
PRTBYT          MOVE D1,D0              ;Kopie nach D0
                LSR #4,D0               ;High-Nibble ins Low-Nibble schieben
                BSR PRTDIG              ;und als ASCII Zeichen drucken
                MOVE D1,D0              ;Byte wieder nach D0 kopieren, und
;
; Nibble in D0 als hexadezimales Zeichen ausgeben
;-----------------------------------------------
PRTDIG          AND #$F,D0              ;oberes Nibble loeschen
                ADDI #$30,D0            ;Offset fuer Zeichen '0' addieren
                CMP.B #$3A,D0           ;falls Zeichen < 'A'
                BCS PRTDIG1             ;sofort ausgeben,
                ADDQ #$7,D0             ;sonst Offset fuer Zeichen 'A' addieren
PRTDIG1         BRA ASCIOUT1            ;und ausgeben
;
; Adresse in A0 hexadezimal ausgeben (6-stellig)
;-----------------------------------------------
PRTADR          MOVE.L A0,D1            ;Kopie nach D1
                SWAP D1                 ;High-Wort ins Low-Wort kopieren
                BSR PRTBYT              ;aber davon nur das Low-Byte ausgeben
                SWAP D1                 ;zaubere wieder das alte Wort herbei
                BRA PRTWORD             ;und gib es aus.
;
; Inhalt von D1 als 2 ASCII-Zeichen ausgeben
;-----------------------------------------
PRT2ASC         MOVE D1,-(A7)           ;Zeichen auf dem Stack sichern
                LSR #8,D1               ;High-Byte ins Low-Byte kopieren
                BSR PRTASC              ;und ausgeben
                MOVE (A7)+,D1           ;Zeichen wieder vom Stack holen und
;
; Inhalt von D1 als ASCII-Zeichen ausgeben (Steuerzeichen in '.' verwandeln!)
;-----------------------------------------------------------------------------
PRTASC          MOVE D1,D0              ;Kopie nach D0
                ANDI #$7F,D0            ;MSB loeschen
                CMP.B #$7F,D0           ;Del nicht ausgeben
                BEQ MAKEP
```

```
                    CMP.B #SPC,D0          ;handelt es sich um druckbares Zeichen?
                    BCC ASCIOUT2           ;dann ausgeben
MAKEP               MOVEQ #'.',D0          ; sonst einen Punkt als Ersatz drucken
ASCIOUT2            BRA ASCIOUT
;
; DUMP REGISTER SELECTED BY MASK IN D1
;
PRTREG              TST.B D1               ;DATA REG SELECTED?
                    BEQ PRTREG1            ; NO
                    MOVEQ #'D',D0
                    BSR ASCIOUT2           ;PRINT "D"
                    MOVEA.L #REGSAV,A4     ;POINTER TO DREG STORAGE
                    BRA PRTREG2
PRTREG1             MOVEQ #'A',D0
                    BSR ASCIOUT2           ;PRINT "A"
                    LSR #8,D1              ;SWAP FLAGS TO B0..B7
                    MOVEA.L #REGSAV+32,A4  ;POINTER TO AREG STORAGE
PRTREG2             MOVEQ #'0',D0          ;REG NUMBER TO PRINT
PRTREG3             LSR #1,D1              ;SHIFT (REGNUMBER) TIMES
                    BCS PRTREG4
                    ADDQ.L #4,A4           ;NOT READY, INCREMENT POINTER
                    ADDQ #1,D0             ; AND REGNUMBER
                    BRA PRTREG3
PRTREG4             BSR ASCIOUT2           ;PRINT REGNUMBER
                    BSR PRTCOL             ;":"
                    MOVE.L (A4),D2
                    BSR PRTLONG            ; REG CONTENTS
                    MOVEQ #1,D1            ;1+1 SPACES
                    BRA PRTMSP
;
; Drucke das Status-Register
;---------------------------
PRTSR               MOVEA.L #SRDAT,A0      ;Adresse der Register-Tabelle
                    MOVE.W STATSAV,D3      ;gespeichertes SR zurueckholen
                    MOVEQ #7,D2            ;8 Flags ausgeben
PRTSR1              MOVE.B (A0)+,D0        ;Name des Flags
                    BSR ASCIOUT            ;drucken
                    MOVE.B (A0)+,D1        ;Bit-Position des Flags
                    MOVE.W D3,D0           ;SR nach D0 kopieren
                    LSR.W D1,D0            ;bis zum gesuchten Bit nach rechts sc
                    AND.W (A0)+,D0         ;obere Bits loeschen
                    BSR PRTDIG             ;Inhalt drucken
                    BSR PRTSPC             ;Leerzeichen
                    DBF                    D2,PRTSR1
;naechstes Flag
                    RTS
;
; Status Register nach Benutzerangabe setzen
;-------------------------------------------
SETSR               MOVEA.L  #SRDAT,A0     ;Adresse der Register-Tabelle
                    MOVEQ    #7,D2         ;es gibt 7+1 Flags
SETSR1              CMP.B    (A0)+,D0      ;ist es dieserFlag-Name?
```

```
                BEQ     SETSR2          ;dann bearbeiten
                ADDQ.L  #3,A0           ;sonst Position usw. uebergehen
                DBF     D2,SETSR1       ;bis zum Ende der Tabelle
                ORI     #1,CCR          ;Carry als Fehlerflag setzen
                RTS
SETSR2          BSR.L   GETNOSP         ;Wert des Flags
                CMP.B   #':',D0         ;ist es ein ':'?
                BNE     SETSR3          ;nein, dann ist eine Zahl
                BSR.L   GETNOSP         ;sonst ':' ignorieren
SETSR3          SUBI.B  #'0',D0         ;ASCII ▸ Digit umwandeln
                MOVE.B  (A0)+,D1        ;Bit-Position des Flags
                MOVE.W  (A0)+,D2        ;und Maximalwert laden
                CMP.B   D0,D2           ;Wert im zulaessigen Bereich?
                BCS     SETSR4          ;nein, dann abbrechen
                LSL.W   D1,D0           ;Wert in richtige Position schieben
                LSL.W   D1,D2           ;Max.-Wert ebenso
                NOT.W   D2              ;invertieren und
                AND.W   D2,STATSAV      ;entsprechende Bits des SR loeschen
                OR.W    D0,STATSAV      ;eingegebene Bits setzen
                ANDI    #$FE,CCR        ;Carry loeschen : ok
SETSR4          RTS
;
; Hole eine Eingabezeile vom Host-Rechner
;-----------------------------------------
GETLINEP        BSR     CROUT           ;Zeilenvorschub
                BSR     PRTCOL          ;Prompt ':' ausgeben
GETLINE         MOVE.L  #BUFFER,A6      ;Zeilenpuffer-Adresse
getlinex        MOVEA.L A6,A5           ;Zeilenende
GETLINE1        BSR.L   TASTIN          ;Zeichen lesen
                CMP.B   #SPC,D0         ;Sonderzeichen?
                BCS     CONTROL         ;dann Sonderbehandlung
                CMPA.L  #BUFFER+78,A5   ;ist der Puffer voll?
                BCC.L   GETLINE3        ;dann geht nichts mehr
                MOVE.B  D0,(A6)+        ;sonst speichern
                BSR     ASCIOUT         ;und ausgeben
                CMPA.L  A5,A6           ;Hinter Zeilenende-Zeiger?
                BLS     GETLINE1        ;nein, dann weiter
                ADDQ.L  #1,A5           ;sonst Ende-Zeiger erhoehen
                BRA     GETLINE1        ;und naechstes Zeichen lesen
CONTROL         CMP.B   #CR,D0          ;Return?
                BEQ     LINEEND
                CMP.B   #BKSPC,D0       ;Linkspfeil?
                BEQ     BACKSPC
                CMP.B   #RAROW,D0       ;Rechtspfeil?
                BEQ     RIGHT
                CMP.B   #TRUNC,D0       ;Zeilenabbruch?
                BEQ     LINEBRK
                BRA     GETLINE3        ;mehr Steuerzeichen gibts nicht!
BACKSPC         CMPA.L  #BUFFER,A6      ;Cursor am Zeilenanfang?
                BEQ     GETLINE3        ;da ist eine Wand!
                SUBQ.L  #1,A6           ;sonst Cursor einen nach links
                BSR     ASCIOUT         ;und Linkspfeil ausgeben
```

```
                BRA     GETLINE1        ;weiter
RIGHT           CMPA.L  A5,A6           ;Cursor rechts vom Zeilenende?
                BCC     GETLINE3        ;da gibts nichts zu kopieren
                MOVE.B  (A6)+,D1        ;sonst alten Text ausgeben
                BSR     PRTASC
                BRA     GETLINE1        ;weiter
GETLINE3        BSR     BELL ;          Piep das war nix!
                BRA     GETLINE1        ;noch einmal bitte
LINEBRK         MOVEA.L A6,A5           ;Zeilenende = akt. Cursorposition
LINEEND         MOVE.B  D0,(A6)+        ;CR speichern
                BSR     CROUT           ;Zeilenvorschub
                MOVEA.L #BUFFER,A6      ;A6 = Zeilenanfang
                RTS
;
; Hole das naechste Zeichen aus dem Zeilenpuffer, Leerzeichen ignorieren
;-----------------------------------------------------------------------
GETNOSP         MOVE.B  (A6)+,D0        ;Hole Zeichen aus dem Puffer
                CMP.B   #SPC,D0         ;ist es ein Leerzeichen?
                BEQ     GETNOSP         ;dann das naechste her!
GROSSKL         CMPI.B  #$60,D0         ;ist es etwa ein Kleinbuchstabe?
                BLE     GKRET           ;nein, dann ok
                SUBI.B  #$20,D0         ;sonst in Grossbuchstaben umwandeln
GKRET           RTS
;
; Hole ein ASCII oder HEX Zeichen aus dem Puffer
;-----------------------------------------------
GETWORD         BTST    #4,D6           ;FOUND CR ALREADY?
                BNE     GETWORD5        ;YES
                MOVE.B  (A6)+,D0        ;GET CHAR FROM BUFFER
                CMP.B   #CR,D0
                BEQ     GETWORD5        ;CR, DONE
                BTST    #2,D6           ;ASCII MODE SELECTED?
                BNE     GETWORD2        ; YES
                CMP.B   #SPC,D0         ;HEX MODE: IGNORE SPACES
                BEQ     GETWORD
                CMP.B   #'"',D0         ;START OF ASCII MODE?
                BNE     GETWORD1        ; NO
                BSET    #2,D6           ; YES, SET ASCII FLAG
                BRA     GETWORD         ;AND GET FIRST CHAR
GETWORD1        BSR     HEXTST          ;HEX DIGIT IN D0?
                BCC     SUBERR          ; NO, ERROR
                CLR.L   D2
GETWORDW        LSL     #4,D2
                EOR     D1,D2
                MOVE.B  (A6)+,D0
                BSR     HEXTST
                BCS     GETWORDW
                BCC     GETWORD3
GETWORD2        MOVE    D0,D2           ;ASCII: PASS CHAR TO D2
                CMP.B   #'"',D0         ;END OF ASCII STRING?
                BNE     GETWORD3        ; NO, DONE
                BCLR    #2,D6           ; YES, SET HEX MODE
```

```
                BRA     GETWORD
GETWORD3        BSET    #3,D6           ;FLAG: NOT EOL
                CMP.B   #CR,D0          ;FOUND CR AFTER LAST HEX DIGIT?
                BNE     GETWORD4        ; NO
                BSET    #4,D6           ; YES, FLAG IT
GETWORD4        RTS
GETWORD5        BCLR    #3,D6           ;NO VALID DATA IN D0
                RTS
;
; TEST D0.B FOR HEX DIGIT, CALC DIGIT INTO D1.B
;
GNSHEXT         BSR     GETNOSP
;
;HexTst prueft, ob ein Buchstabe ein Hex-Wert ist
;
HEXTST          BSR     GROSSKL         ;evtl. Kleinbuchst. konvertieren
                MOVEQ   #'0',D1         ;Test, ob Ziffer
                EOR     D0,D1
                CMP     #10,D1
                BCS     HEXTST1         ;D0 IN 0..9 OK
                SUB     #$71,D1         ; A..F Test
                CMP     #$06,D1
                BCC     HEXTST1         ;D0 nicht in A..F
                ADDI.B  #10,D1          ;konvertieren zum alten Wert
                ORI     #1,SR           ;Set Carry fuer Hextest-Fehler
HEXTST1         RTS
;
; SET UP REG MASK
;
GETREG          MOVEQ   #1,D2           ;SET UP REG MASK FOR D0
                CMP.B   #'D',D0         ;DATA REG?
                BEQ     GETREG2         ; YES
                CMP.B   #'A',D0         ; NO, ADR REG?
                BEQ     GETREG1         ;YES
                CMP.B   #'S',D0         ;STATUS REG?
                BNE     GETREG3         ; NO, ERROR
                BSR     GETNOSP
                CMP.B   #'R',D0
                BNE     GETREG3         ;NEXT CHAR MUST BE 'R'
                MOVE.L  #$10000,D2      ;FLAG BIT FOR SR
                ANDI    #$FE,CCR        ;CLEAR CARRY
                RTS                     ;DONE
GETREG1         MOVE    #100,D2         ;REG MASK FOR A0
GETREG2         BSR     GETNOSP         ;GET REG NUMBER
                EORI.B  #'0',D0         ;CONVERT TO HEX
                CMP.B   #8,D0           ;NUMBER > 7 ?
                BCC     GETREG3         ; YES
                LSL     D0,D2           ; NO, SHIFT MASK
                RTS
GETREG3         ORI     #1,SR           ;ERROR: SET CARRY
                RTS
;
```

```
; SET UP REGFLAG FOR LIST OF REGISTERS
;
REGLIST         MOVEQ   #0,D3           ;CLEAR RESULT
                MOVEQ   #1,D2           ;DEFAULT START: D0
REGLIST1        CMP.B   #CR,D0          ;END OF LINE?
                BEQ     REGLIST5        ; YES
                CMPI.B  #$2D,D0         ; '-'-Zeichen? RANGE OF REGS?
                BEQ     REGLIST2        ; YES
                BSR     GETREG          ;GET SINGLE REG MASK
                BCS     SUBERR          ; ERROR
                OR.L    D2,D3           ;ASSEMBLE FLAG
                BRA     REGLIST4        ;NEXT REG
REGLIST2        BSR     GETNOSP         ;REGISTER RANGE:
                MOVE.W  D2,D1           ;SAVE START REG MASK
                BEQ     SUBERR          ;SR SPECIFIED WITH RANGE = ERROR
                BSR     GETREG          ;GET END REG
                BCS     SUBERR          ; ERROR
                TST.W   D2              ;SR SPECIFIED AS END OF RANGE?
                BEQ     SUBERR          ; YES, ILLEGAL!
REGLIST3        OR.W    D1,D3           ;ASSEMBLE FLAG
                CMP     D1,D2           ;REACHED END REG?
                BEQ     REGLIST4        ; YES
                LSL.W   #1,D1           ; NO, NEXT REG MASK
                BCC     REGLIST3        ; UNTIL A7 REACHED
REGLIST4        BSR     GETNOSP
                CMP.B   #'/',D0         ;'/' DELIMITER IS OPTIONAL
                BEQ     REGLIST4        ; IGNORE IT
                BRA     REGLIST1
REGLIST5        RTS
;
;D2 Longword aus Eingabezeile lesen
;D1 muss den ersten Buchstaben bereits enthalten
;
GETLONG         MOVEQ   #0,D2           ;CLEAR REG
GETLONG1        LSL.L   #4,D2           ;SHIFT OLD DIGITS
                EOR     D1,D2           ;ADD NEW DIGIT
                BSR     GNSHEXT         ;GET NEXT CHAR
                BCS     GETLONG1        ; OK, CONTINUE
GETLONG2        RTS                     ; END OF HEX NUMBER
;
MELDUNG         str     'Diskfehler: '
UNGSTR          str     'Fehler: ungerade Adresse'
                sync

QUIT            MOVEQ   #11,D2
                BSR.L   CODEOUT
                move    #50000,d0
quitpause       dbf     d0,quitpause
                BRA     KaltSt

screeninfo      clr     d2
                bsr.l   codeout ;screeninfo lesen
```

```
                    move    #$0702,bildgr
                    bsr     bytein
                    move.b  d0,xmax
                    cmpi.b  #79,d0
                    blt     kleinbild
                    move    #$0f05,bildgr
kleinbild           bsr     bytein
                    move.b  d0,ymax
                    rts
CloseOffen          move    FileFlags,D3    ;Schliesst alle offenen Files
                    move    #$F,D4
closemehr           btst    d4,d3
                    BEQ     Nichtoffen
                    move    d4,d1
                    bsr.l   close
nichtoffen          subq    #1,d4
                    bpl     closemehr
                    rts

BUSSTR              str     'Busfehler-Zugriff auf: '
ERRSTR1             str     'Falscher Zugriff auf '
ERRSTR2             str     'Unbekannte Anweisung in '

anftext             str     '****KAT-Ce Version 1.3 ****'
ramendtext          str     'RAM-Endebei $'
textanftext         str     'Editortext ab $'
codanftext          str     'Assembler Code ab $'
wastuntext          str     'E(ditor W(arm-Editor M(onitor'
wasnochtext         str     'X(ecute A(ssembler R(un Q(uit'
KeinRCText          str     '»»» Kein Code verfuegbar'
XFileStr            str     'Welches File starten: '
                    SYNC
;
; COLDSTART OF MONITOR
;
;
MONINIT             MOVEM.L D0-A7,REGSAV    ;alle Register speichern
MONIN               MOVE.L  #0,A1     ;Startwert fuer Hex-Adressen setzen
                    MOVE.L  A1,A2     ;Startwert fuer Endadresse
                    MOVE.L  #$0001FFFF,REGFLAG      ;zunaechst alle Register zeigen
                    BSR.L   CROUT
                    CLR.L   BUFFER
                    CLR.W   MONFLAGS

;
; WARMSTART OF MONITOR
;
MON1                MOVE    SR,STATSAV      ;Werte fuer SR speichern

                    ;Begruessung

begruessung         bsr     clbild    ;Bild loeschen
```

```
rien            ;Einstieg  fuer nicht definierte Befehle
begruess2       BSR        CloseOffen
                MOVE.L     *RETURN,OUTVECT;RTS - BEFEHL
                move.l     a7,StartA7
                bsr        crout2
                lea        anftext,a1          ;Text ausgeben
                bsr.l      strout
                bsr        crout2
                lea        Textanftext,a1
                bsr.l      strout
                move.l     Textstart,A0
                bsr        prtadr
                bsr        crout
                lea        codanftext,a1
                bsr.l      strout
                move.l     codestart,a0
                bsr        prtadr
                bsr        crout
                lea        ramendtext,a1
                bsr.l      strout
                move.l     ramende,a0
                subq.l     #1,a0
                bsr        prtadr
                bsr        crout2
                lea        wastuntext,a1
                bsr.l      strout
                bsr        crout
                lea        wasnochtext,a1
                bsr.l      strout
                bsr        screeninfo
wastunwarte     bsr        TASTIN
                bsr        grosskl
                cmpi.b     #'A',d0
                beq.l      AssStart
                cmpi.b     #'X',d0
                beq.l      ExecuteFile
                cmpi.b     #'R',d0
                beq.l      RunRamCode
                cmpi.b     #'E',d0
                beq.l      kateditor
                cmpi.b     #'W',d0
                beq.l      warmedi
                cmpi.b     #'M',d0
                beq.l      mon2
                cmpi.b     #'Q',d0
                beq        quit
                bra        wastunwarte
ExecuteFile     bsr        crout2
                lea        XFilestr,a1
                bsr.l      strout
                move.l     #buffer,a6          ;Platz fuer Laengenbyte
                move.b     #'  ',(a6)+         ;des Filenamens schaffen
```

```
                bsr     getlineX        ;Name des Files holen
                move.b  #',',(a5)+      ;Start des Files ermoeglichen
                move.b  #'G',(a5)+
                bsr.l   lesen           ;File einlesen und ausfuehren
                bra     begruess2
RunRamCode      bsr     crout2
                move.l  CodeStart,A0
                move.l  A0,D0
                addq.l  #8,D0
                cmp.l   (a0),d0
                beq     lasslaufen
                lea     KeinRCText,a1
                bsr.l   strout
                bra     begruess2
lasslaufen      move.l  d0,a0
                jsr     (a0)
                bra     begruess2
;
; MONITOR MAIN LOOP
;
MON2            CLR     D6              ;CLEAR OPERATING FLAGS
                ANDI    #$7FFF,STATSAV  ;NOTRACE
                BSR     GETLINEP        ;GET CMD FROM USER
                BSR     cleoline
                BSR     GNSHEXT
                BCC     MON21           ; NO
                BSR     GETLONG         ; YES, GET START ADDRESS
                BCLR    #0,D2
MON44           MOVEA.L D2,A1
                MOVEA.L D2,A2           ;END ADR DEFAULTS TO START
                BSET    #0,D6           ;FLAG: START ADR GIVEN
MON21           CMP     #$2E,D0         ;NEXT CHAR = '.'?
                BNE     MON22           ; NO
                BSR     GNSHEXT
                BCC     MON22           ;NO
                BSR     GETLONG         ;YES, GET END ADDRESS
                BCLR    #0,D2           ;gerade machen
                MOVEA.L D2,A2
                BSET    #1,D6           ;FLAG: END ADR GIVEN
MON22           MOVEQ   #CMDEND-1-CMDLST,D1
                MOVEA.L #CMDEND,A0
MON23           CMP.B   -(A0),D0        ;SEARCH FOR CMD CHAR
                BEQ     MON24           ; FOUND
                DBF     D1,MON23
                BSR     ERROR           ;INVALID CMD CHAR
                BRA     MON2
MON24           LSL     #1,D1           ;INDEX TO JMP TABLE
                MOVE.L  #JMPLST,A0
                MOVE    $0(A0,D1),-(A7) ;GET ADDRESS OF CMD ROUTINE
                MOVE.L  #STARTADR,D1
                MOVE    (A7)+,D1
                MOVE.L  D1,A0
```

```
                JSR     (A0)            ;EXECUTE CMD
                BRA     MON2            ;GET NEXT CMD
;
; TRAP #0 schaltet TRACE aus:
;-----------------------------
TraceAus        andi.w  #$7FFF,(a7)     ;
                andi.b  #$1F,MonFlags
                andi    #$7FFF,StatSav
                RTE
;
;TRAP #1 schaltet TRACE ein:
;
TraceEin        ori.w   #$8000,(a7)
                ORI     #$8000,STATSAV  ;Trace Mode setzen
                ori.b   #$E0,monflags
;
; EXECUTE TRACE EXCEPTION: DUMP PC AND REGISTERS
;
;
;
TraceIt         ADDQ.L  #6,A7
                MOVEM.L D0-A7,REGSAV    ;Registerwerte speichern
                SUBQ.L  #6,A7
TraceIt7        move    (a7),statsav    ;Statusregister lesen
                BTST    #6,MONFLAGS     ;Startbit gesetzt?
                BNE     TraceIt4        ;ja, zunaechst den ersten Befehl
                                        ;ausfuehren
                MOVE.L  ZNUMMER,A1      ;Adresse der zu disass. Zeile
                BSR.L   DISZEILE        ;Zeile Disass
                MOVE.L  REGFLAG,D3      ;Registerliste ausgeben
                BSR.L   REGDSP1
                BSR     CROUT
                CMPI.L  #BACK1,2(A7)    ;Ist der Ausgangszustand des
                                        ;Stacks erreicht?
                BEQ     TraceItRET      ;ja, Trace-Ende
                BTST    #7,MONFLAGS     ;Step Mode?
                BNE     TraceIt3        ;ja
TraceIt2        BSR     KEYPRESS        ;Laufmodus: Taste gedrueckt?
                TST.B   D0
                BEQ     TraceIt4        ;nein
TraceIt3        BSR     TASTIN          ;Taste lesen
                CMP.B   #SPC,D0         ;Leertaste?
                BNE     TraceIt5        ;nein
                BSET    #7,MONFLAGS     ;StepModus setzen
                BRA     TraceIt4        ;fertig
TraceIt5        CMP.B   #CR,D0          ;RET gedrueckt?
                BNE     TraceIt6        ;nein
                BCLR    #7,MONFLAGS     ;Laufmodus setzen
                BRA     TraceIt4        ;fertig
TraceIt6        CMP.B   #$03,D0         ;CTRL-C gedrueckt?
                BNE     TraceIt4        ;nein
TraceItR        btst    #5,monflags     ;Trace durch TRAP-Befehl eingeschaltet?
```

```
                bne     traptrende      ;ja
                CMPI.L  #BACK1,2(A7)    ;Routine beendet?
                BEQ     TraceItRET      ;Trace - Ende
                ADDQ.L  #2,A7           ;Stack wiederherstellen
                BRA     TraceItR        ;fertig
TraceIt4        MOVE.L  2(A7),ZNUMMER   ;naechste Adresse fuer Disass. speichern
                BCLR    #6,MONFLAGS     ;Startflag loeschen
                MOVEM.L REGSAV,D0-A6    ;Register wiederherstellen
                RTE                     ;das wars
TraceItRET      ADDQ.L  #6,A7           ;Stack korrigieren
                ANDI    #$7FFF,SR       ;Trace loeschen
                ANDI    #$7FFF,Statsav  ;auch im Speicher
                BRA.L   BACK1           ;Sprung zum Ende der Routine
traptrende      andi    #$7FFF,(a7)     ;Trace loeschen
                andi.b  #$1F,monflags   ;auch im Speicher
                bra     TraceIt4
;
;
;
; EVALUATE COUNT IF SPECIFIED
;
RANGE           BSR.L   GNSHEXT         ;lies Zeichen
                BCC     RANGE2          ;kein Hex
                BSR     GETLONG         ;lies Longword
                SUBQ.L  #1,D2
                BTST    #1,D6           ;ex. Endadresse?
                BNE     RANGE1          ;ja
                LSL.L   #1,D2
                ADD.L   A1,D2           ;nein END:=START+COUNT
                MOVE.L  D2,A2
                BRA     RANGE2
RANGE1          BTST    #0,D6           ;ex. auch Startadresse?
                BNE     RANGE2          ;ja, Zaehler ignorieren
                MOVEA.L A2,A1           ;nein START:=END-COUNT
                SUBA.L  D2,A1
RANGE2          RTS
;
; MOVE CMD
;
MOVE            BSR.L   GETLTGER        ;gerade Adresse lesen
                MOVEA.L D2,A3
                MOVEA.L A1,A4           ;Quellenzeiger
                MOVE.L  A2,D3           ;Ende der Quelle
MOVE1           MOVE.B  (A4)+,(A3)+     ;kopieren
                CMP.L   A4,D3           ;fertig?
                BCC     MOVE1           ;nein
                RTS
;
; GO CMD
;
GO              MOVE    A1,D0
                BTST    #0,D0           ;gerade Adresse?
```

```
                BNE     ERROR           ;nein, Fehler
GO1             MOVEM.L A1-A3/A6,-(A7)  ;Monitor-Register retten
                MOVE.L  A1,A6           ;Start Adresse
                MOVE.B  D6,MONFLAGS     ;SAVE S/T FLAG
                MOVEM.L REGSAV,D0-A5    ;Register laden
                MOVE    STATSAV,SR      ;SR auch
                JSR     (A6)            ;Programm starten
BACK1           MOVE    SR,STATSAV      ;SR und
                MOVEM.L D0-A7,REGSAV    ;Registerinhalte speichern
                MOVEM.L (A7)+,A1-A3/A6  ;Monitor-Register wieder her
                MOVE.B  MONFLAGS,D6     ;D6-Flags wiederherstellen
                RTS
;
SETMEM          BSR     GETWORD         ;holo Daten
                BTST    #3,D6           ;gueltige Daten?
                BEQ     SETMEM1         ;nein
                BTST    #2,D6           ;HEXMODE?
                BEQ     SETHEX          ;ja
                MOVE.B  D2,(A1)+        ;direkt speichern
                BRA     SETMEM
SETHEX          MOVE    A1,D0           ;gerade Adresse?
                AND     #1,D0
                BEQ     ADRGER          ;ja
                MOVE.B  #0,(A1)+        ;nein, 1 Byte speichern
ADRGER          MOVE.W  D2,(A1)+        ;speichern
                BRA     SETMEM          ;weitermachen
SETMEM1         RTS
;
;
TASTE           BSR     KEYPRESS        ;Taste gedrueckt?
                TST.B   D0
                BEQ     TASTRET         ;nein
                BSR     TASTIN          ;Taste lesen
                CMPI.B  #$3,D0          ;CONTR C?
                BEQ     TASTE2          ;ja
                BSR     TASTIN          ;noch eine Taste lesen
                CMPI.B  #$3,D0          ;CONTR C?
                BEQ     TASTE2          ;ja
TASTRET         RTS
TASTE2          ADDQ.L  #4,A7           ;'PULL'
                RTS
;
; VERIFY CMD
;
VERIFY          BSR.L   GETLTGER
                MOVEA.L D2,A3
                MOVEA.L A1,A4           ;SET UP SOURCE POINTER
                MOVE.L  A2,D3
VERIFY1         MOVE.W  (A4),D1
                CMP.W   (A3),D1         ;COMPARE BYTES
                BEQ     VERIFY2
                MOVEA.L A4,A0           ;NOT EQUAL:
```

```
                BSR     PRTADR          ;PRINT SOURCE ADR
                BSR     PRTCOL;':'
                MOVE.W  (A4),D1
                BSR     PRTWORD         ;SOURCE BYTE
                MOVEQ   #2,D1           ;3 SPACES
                BSR     PRTMSP
                MOVEA.L A3,A0
                BSR     PRTADR          ;DEST ADR
                BSR     PRTCOL          ;':'
                MOVE.W  (A3),D1
                BSR     PRTWORD         ; DEST BYTE
                BSR     CROUT           ;CR
                BSR     TASTE
VERIFY2         CMPM.W  (A4)+,(A3)+     ;INCREMENT POINTERS
                CMP.L   A4,D3           ;REACHED END OF RANGE?
                BCC     VERIFY1         ; NO
                RTS
;
; EXAMINE MEMORY CMD
;
XAMINE          BSR     RANGE           ;EVALUATE OPTIONAL COUNT
XAMINE1         move.l  a1,d1           ;ungerade Adresse verhindern
                bclr    #0,d1
                move.l  d1,a1
                MOVE.L  A1,A0           ;START ADDRESS
                MOVE.L  A1,A4
                BSR     PRTADR          ;PRINT ADR
                BSR     PRTCOL          ;':'
XAMINE2         BSR     PRTSPC
                MOVE.W  (A1)+,D1
                BSR     PRTWORD         ;PRINT BYTE IN HEX
                MOVE.L  A1,D2
                CMP.L   A2,D2           ;END OF RANGE?
                BHI     AASC            ; YES
                AND.B   BILDGR,D2       ;END OF SCREEN LINE?
                BNE     XAMINE2         ; NO
AASC            CLR.L   D1
                MOVE.B  BILDGR,D1
                LSL     #2,D1
                SUB.B   XKOORD,D1
                BSR     PRTMSP
AASCW           MOVE.W  (A4)+,D1
                BSR     PRT2ASC
                MOVE.L  A4,D2
                CMP.L   A2,D2
                BHI     CROUT
                AND.B   BILDGR,D2
                BNE     AASCW
                BSR     CROUT
                BSR     TASTE
                BRA     XAMINE1
;
```

```
; SET/DISPLAY REGISTERS
;
REGDSP      BSR     GETNOSP
            CMP.B   #'=',D0         ;NEXT CHAR = '='?
            BEQ     REGSET          ; YES, SET REGISTER CONTENTS
            BSR     REGLIST         ; NO, GET REG LIST
            TST.L   D3              ;ANY REG'S SPECIFIED?
            BNE     REGDSP1
            MOVE.L  REGFLAG,D3      ; NO, USE DEFAULT
REGDSP1     MOVEQ   #1,D4
            MOVEQ   #0,D5           ;START IN FIRST COLUMN
REGDSP2     MOVE.W  D4,D1
            AND.W   D3,D1           ;THIS REG SPECIFIED?
            BEQ     REGDSP3         ; NO
            DBF     D5,REGDSP5      ;COUNT DOWN COLUMNS
            BSR     CROUT           ;START NEW LINE
            CLR.L   D5
            MOVE.B  BILDGR+1,D5     ;RESET COUNTER: ANZAHL SPALTEN
REGDSP5     BSR     PRTREG          ; YES, PRINT IT
REGDSP3     LSL.W   #1,D4           ;SHIFT TO NEXT MASK
            BCC     REGDSP2         ; UNTIL A7 REACHED
            BTST    #$10,D3         ;WHAT ABOUT SR?
            BEQ     REGDSP4         ; NOT SELECTED
            BSR     CROUT
            BSR     PRTSR           ; YES, PRINT SR
REGDSP4     BRA     CROUT
REGSET      BSR     GETNOSP
            BSR     GETREG          ;GET REG MASK
            BCC     REGSET1         ;A OR D REG SPECIFIED
            BSR     SETSR           ;STATUS REG FLAGS SPECIFIED?
            BCS     ERROR           ; NO
            BSR     GETNOSP         ;GET NEXT CHAR
            BRA     REGSET4         ;NEXT REGISTER
REGSET1     TST.W   D2              ;SR SPECIFIED?
            BEQ     ERROR           ; YES
            MOVEA.L #REGSAV,A4      ;POINTER TO DREG STORAGE
            TST.B   D2              ;DREG SPECIFIED?
            BNE     REGSET2         ; YES
            LSR     #8,D2           ; NO, SWAP FLAGS
            MOVEA.L #REGSAV+32,A4   ;POINTER TO AREG STORAGE
REGSET2     LSR     #1,D2
            BCS     REGSET3
            ADDQ.L  #4,A4           ;ADJUST POINTER ACC. TO REG NUMBER
            BRA     REGSET2
REGSET3     BSR     GETNOSP
            CMP.B   #':',D0         ;NEXT CHAR = ':'?
            BNE     ERROR           ; NO
            BSR     GETLTEST
            MOVE.L  D2,(A4)         ;SET UP IN STORAGE AREA
REGSET4     CMP.B   #'/',D0         ;DELIMITER FOLLOWING?
            BEQ     REGSET          ; YES, PROCEED
            CMP.B   #CR,D0          ; NO, END OF LINE?
```

```
                BNE     ERROR           ;NO
                RTS
;
; REGISTER OUTPUT CONTROL
;
OUTPUT          BSR     GETNOSP
                BSR     REGLIST         ;GET REGISTER LIST
                MOVE.L  D3,REGFLAG      ;STORE
                RTS
;
; TRACE CMD
;
TRACE           ori.b   #$C0,D6         ;Single Step Modus, Startbit setzen
                ORI     #$8000,STATSAV  ;im Speicher Trace-Bit setzen
                BRA     GO              ;Starte Programm
;
SUCHERR         MOVEQ   #0,D1
                ADDQ.L  #8,A7
                RTE
;
;
ADRERR          MOVEA.L #ERRSTR1,A1
                BRA     ERROUT
ILLINSTR        MOVEA.L #ERRSTR2,A1
ERROUT          BSR     CROUT
                BSR.L   STROUT
                MOVE.L  2(A7),A0
                BSR     PRTADR
                BSR     CROUT
                BSR     BELL

                move.l  startA7,A7
                jmp     begruess2
;
;LONG SUCHEN
;
GETLTEST        BSR     GETNOSP
                BSR     HEXTST
                BCC     SUBERR
                BRA     GETLONG
GETLTGER        BSR     GETLTEST
                BCLR    #0,D2
                RTS
;
indiskbuffer    movem.l A1/D1,-(A7)     ;schiebt aus (A0) in Diskbuffer
                move    #512,d1
                lea     diskb,a1
indbweiter      move.b  (a0)+,(a1)+
                subq    #1,d1
                bne     indbweiter
                movem.l (A7)+,A1/D1
                rts
```

```
ausdiskbuffer   movem.l  A1/D1,-(A7)      ;schiebt Diskbuffer nach (A0)
                move     #512,d1
                lea      diskb,a1
ausdbweiter     move.b   (a1)+,(a0)+
                subq     #1,d1
                bne      ausdbweiter
                movem.l  (A7)+,A1/D1
                rts

;
SCHREIBE        BSR.L    Name
                BSR      GETLTGER
                MOVE.L   D2,D3
                BSR      GETLTGER
                MOVE.L   D2,D4           ;LAENGE SPEICHERN
;
; SENDEVORGANG
;
                CLR.L    D1
                BSR.I    REWRITE
                BSR.I    AlfredIORes
                CLR.L    D5
                MOVEA.L  D3,A1
                MOVEA.L  #DISKB,A0
                MOVE.L   D3,(A0)+        ;ANFANG
                MOVE.L   D4,(A0)+        ;LAENGE DES PROGRAMMS
                ADDI.L   #8,D4
                MOVEA.L  D3,A1           ;ANFANG
BFILL           MOVE.L   (A1)+,(A0)+
                CMPA.L   #DISKB+512,A0   ;BUFFER VOLL?
                BNE      BFILL
                CLR.L    D1
                BSR.I    BLOCKW
                BSR.I    AlfredIORes
                ADD.L    #1,D5
                MOVEA.L  #DISKB,A0
                SUBI.L   #512,D4
                BGT      BFILL
                CLR.L    D1
                BSR.L    CLOSE
                BRA.L    AlfredIORes
;
LESEN           BSR.L    NAME
                bclr     #0,monflags     ;kein Go
                clr.l    D2              ;zeigt an, ob neue Startadresse
                cmpi.b   #$D,D0          ;Zeilenende?
                beq      nolesestart     ;ja
                cmpi.b   #',',D0         ;Komma?
                BNE      nolesestart     ;nein
                BSR      GNSHEXT         ;folgt ein HexZeichen?
                BCC      testgo          ;nein
                BSR      GETLONG         ;Startadresse in D2 einlesen
```

```
                cmpi.b   #',',D0        ;folgt Komma?
                bne      neulesestart   ;nein
                MOVE.B   (A6),D0
                BSR      grosskl
                CMPI.B   #'G',D0
                BNE      neulesestart
                BSET     #0,monflags
                BRA      neulesestart
testgo          cmpi.b   #'G',D0
                bne      nolesestart
                bset     #0,monflags    ;Go anzeigen
nolesestart     clr.l    D2             ;zeigt an, ob neue Startadresse
neulesestart    MOVE.L   D2,D7          ;evtl Startadresse speichern
                CLR.L    D1             ;Filenummer
                BSR.L    RESETF
                BSR.L    AlfredIORes    ;Test auf Diskfehler
                CLR.L    D5             ;Blockzaehler
                CLR.L    D1
                BSR.L    BLOCKR
                BSR.L    AlfredIORes
                MOVEA.L  #DISKB,A0
                MOVEA.L  (A0)+,A1       ;Anfang lesen
                MOVE.L   A1,D3          ;speichern
                TST.L    D7             ;evtl neue Startadresse
                BEQ      noneustart     ;nicht gewuenscht
                MOVE.L   D7,D3          ;ja, setzen
                MOVE.L   D7,A1
noneustart      MOVE.L   (A0)+,D4       ;Laenge
LEEREN          MOVE.L   (A0)+,(A1)+
                SUBQ.L   #4,D4
                BLS      FERTIG
                CMPA.L   #DISKB+512,A0
                BNE      LEEREN
                ADDQ.L   #1,D5
                CLR.L    D1
                BSR.L    BLOCKR
                BSR.L    AlfredIORes
                MOVEA.L  #DISKB,A0
                BRA      LEEREN
FERTIG          CLR.L    D1
                BSR.L    CLOSE
                BSR      AlfredIORes
                BTST     #0,Monflags
                BNE      laufen
RET             MOVEA.L  #LSTR,A1
                BSR.L    STROUT
                MOVEA.L  D3,A0
                BRA      PRTADR
LSTR            str      'Anfang bei: '
                sync

;
LAUFEN          MOVEA.L  D3,A2
```

```
                JMP     (A2)
;
;
NAME            BSR     GETNOSP         ;ersten Buchstaben suchen
                MOVEA.L A6,A1           ;Zeiger auf Anfang+1
                SUBQ.L  #1,A1           ;A1 zeigt auf 1. Buchstaben
NAME1           MOVE.B  (A6)+,D0        ;Zeichen lesen
                CMP.B   #$D,D0          ;Zeile fertig?
                BEQ     NAME3           ;ja
                CMP.B   #',',D0         ;Komma?
                BNE     NAME1           ;nein, weitermachen
NAME3           MOVE.L  A6,D2           ;Laenge berechnen
                SUB.L   A1,D2
                SUBQ.L  #1,D2
                MOVE.B  D2,-(A1)        ;Laengenbyte setzen
                RTS
;
REWRITE         BSR     SetFileFlag
                MOVEQ   #6,D2
                BSR.L   CODEOUT
                BSR     FNUMMEROUT
                BRA.L   CSTROUT

;GOTOXY D1 ENTHAELT XKOORD IM LOWBYTE, YKOORD IM HIGHBYTE

GOTOXY          MOVEQ   #10,D2
                BSR.L   CODEOUT
                MOVE    D1,D0
                MOVE.B  D0,XKOORD
                BSR     BYTEOUT
                MOVE    D1,D0
                LSR     #8,D0
                MOVE.B  D0,YKOORD
                BRA     BYTEOUT

AlfredIORes     movem.l d0-d2,-(a7)
                BSR     IORESULT
                tst     d0
                BEQ     NOERR
                BSR.L   IOERR
                move.l  StartA7,A7      ;Stackpointer-korrektur
                BRA     begruess2
NOERR           movem.l (a7)+,d0-d2
                RTS

SetFileFlag     move    FileFlags,D2
                bset    D1,D2
                move    D2,FileFlags
                RTS

ClearFileFlag   move    FileFlags,D2
                bclr    D1,D2
```

```
                move    D2,FileFlags
                RTS

IORESULT        MOVEQ   #9,D2
                BSR.S   CODEOUT
                BRA     BYTEIN

FNUMMEROUT      MOVE    D1,D0
                BRA     BYTEOUT

CLOSE           BSR     ClearFileFlag
                MOVEQ   #7,D2
                BSR.S   CODEOUT
                BRA     FNUMMEROUT

CLOSEDEL        BSR     ClearFileFlag
                MOVEQ   #8,D2
                BSR.S   CODEOUT
                BRA     FNUMMEROUT

RESETF          BSR     SetFileFlag
                MOVEQ   #5,D2
                BSR.S   CODEOUT
                BSR     FNUMMEROUT
                BRA.L   CSTROUT

CODEOUT         MOVEQ   #DC1,D0
                BSR     BYTEOUT
                MOVE.B  D2,D0
                BRA     BYTEOUT

BLOCKW          MOVEQ   #4,D2
                BSR     CODEOUT
                BSR     FNUMMEROUT
                MOVE.L  D5,D0
                BSR     WORDOUT
                MOVEA.L #DISKB,A0
BLOCKW1         MOVE.B  (A0)+,D0
                BSR     BYTEOUT
                CMPA.L  #DISKB+512,A0
                BNE     BLOCKW1
                RTS
;
BLOCKR          MOVEQ   #3,D2
                BSR     CODEOUT
                BSR     FNUMMEROUT
                MOVE.L  D5,D0
                BSR     WORDOUT
                MOVEA.L #DISKB,A0
BLOCKR1         BSR     BYTEIN
                MOVE.B  D0,(A0)+
                CMPA.L  #DISKB+512,A0
```

```
                BNE     BLOCKR1
                RTS

UNITOUT         MOVEM.L D0/D2,-(A7)
                MOVEQ   #1,D2
                BSR.S   CODEOUT
                movem.l (A7)+,D0/D2
                BRA     BYTEOUT

UNITINIT        movem.l D0/D2,-(A7)
                MOVEQ   #2,D2
                BSR     CODEOUT
                MOVEM.L (A7)+,D0/D2
                RTS

IOERR           MOVE.B  D0,D5           ;FEHLERNUMMER
                BSR     CLOSEdel
ERR19           MOVEA.L #MELDUNG,A1
                BSR     STROUT
                MOVE.B  D5,D1
                BSR     PRTBYT
                BRA     BELL
;
CSTROUT         MOVE.B  (A1),D0
                BSR     BYTEOUT
STROUT          MOVE.B  (A1)+,D1        ;A1 ZEIGER AUF STRING
                beq     stroutrts
W               MOVE.B  (A1)+,D0
                BSR     ASCIOUT
                SUBQ.B  #1,D1
                BNE     W
stroutrts       RTS
;
RECHNEN         BSR     GETLTEST
                MOVE.L  D2,D3
                MOVE.B  -(A6),D4
                MOVE.B  (A6)+,D2
                BSR     GETLTEST
                CMPI.B  #$2B,D4;'+'
                BEQ.S   PLUS
                CMPI.B  #$2D,D4;'-'?
                BEQ.S   MINUS
                CMPI.B  #'*',D4
                BEQ.S   MAL
                CMPI.B  #'/',D4
                BNE     BELL
DIV             TST.L   D2
                BEQ     BELL
                DIVU    D2,D3
                MOVE.W  D3,D1
                MOVEQ   #'=',D0
                BSR     BYTEOUT
```

```
                LSR     #8,D1
                BSR     PRTBYT
                MOVE.B  D3,D1
                BSR     PRTBYT
                MOVEQ   #'&',D0
                BSR     BYTEOUT
                SWAP    D3
                MOVE    D3,D1
                LSR     #8,D1
                BSR     PRTBYT
                MOVE    D3,D1
                BRA     PRTBYT
MAL             MULU    D3,D2
RAUSG           MOVEQ   #'=',D0
                BSR     BYTEOUT
                BRA     PRTLONG
MINUS           SUB.L   D2,D3
                MOVE.L  D3,D2
                BRA     RAUSG
PLUS            ADD.L   D3,D2
                BRA     RAUSG
;
;
AUSGABE         CLR.L   D0
                MOVE.B  (A6)+,D0
                CMP.B   #$30,D0
                BMI     BELL
                CMP.B   #$35,D0
                BPL     BELL
                cmp.b   #$33,d0
                bne     noP3
                move.b  seraBaud,D1
NoP3            cmp.b   #$34,D0
                bne     noP4
                move.b  serbbaud,d1
NoP4            AND     #$7,D0
                LSL     #2,D0
                MOVE.L  #OUTTAB,A0
                MOVE.L  $0(A0,D0),OUTVECT
                MOVE.L  #INITTAB,A0
                MOVE.L  $0(A0,D0),A0
                JMP     (A0)
INITTAB         DC.L    BRETURN
                DC.L    PARAINIT
                DC.L    UNITINIT
                DC.L    SERAINIT
                DC.L    SERBINIT
OUTTAB          DC.L    BRETURN
                DC.L    PARAOUT
                DC.L    UNITOUT
                DC.L    SERAOUT
                DC.L    SERBOUT
```

```
;
; Routinen fuer Ein- Ausgabekanaele
;
;in D0 stehen die Ein- bzw Ausgabebytes
;
;bei den Status-Routinen steht in D0:
;Bit 0 gesetzt: sendebereit
;Bit 1 gesetzt: Byte angekommen

PAUSE           MOVE.L  D0,-(A7)
                MOVEQ   #$FF,D0
PAUSEW          SUBQ.B  #1,D0
                BNE     PAUSEW
                MOVE.L  (A7)+,D0
BReturn         RTS

SERAINIT        MOVE.L  A0,-(A7)        ;A0 retten
                bsr     STimerInit
                bsr     SerInitWeiter
                MOVE.B  #$55,$1D(A0)    ;HANDSHAKEAUSG=BEREIT
                move.l  (a7)+,A0
                RTS
SERBINIT        MOVE.L  A0,-(A7)        ;A0 retten
                bsr     STimerInit
                adda    #$10,A0         ;B-Register erreichen
                bsr     SerInitWeiter
                MOVE.B  #$AA,$D(A0)     ;HANDSHAKEAUSG=BEREIT
                move.l  (a7)+,A0
                RTS

STimerInit      MOVE.L  #$402000,A0     ;Grundadresse des 68681
                                        ;Timer auf 38400 Bd stellen
                MOVE.B  #$E0,$9(A0)     ;Timer arbeitet mit Quarzfrequenz 3,68 Mhz
                MOVE.L  #$00000003,$C(a0);beide Timer-Register setzen
                RTS
SerInitWeiter   MOVE.B  D1,$3(A0)       ;In D1 Baudraten-Code setzen
                MOVE.B  #$45,$5(A0)     ;Reset Error Status
                MOVE.B  #$25,$5(A0)     ;Reset Receiver
                MOVE.B  #$35,$5(A0)     ;Reset Transmitter
                MOVE.B  #$15,$5(A0)     ;Reset Modere. Pointer
                MOVE.B  #$13,$1(A0)     ;8 Bit, no Parity
                                        ;jetzt ist MR2A zugaenglich
                MOVE.B  #$17,$1(A0)     ;CTS enable Transmitter
                                        ;1 Stopbit
                RTS
SERAOUT         move.l  a0,-(a7)        ;A0 retten
                move.l  #$402000,a0     ;Adresse Port A
                bra     zweg
SERBOUT         MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  #$402010,A0     ;Adresse Port B
ZWEG            BTST    #2,$3(A0)       ;Transmitter ready
                BEQ     ZWEG            ;nein
```

```
                MOVE.B  D0,$7(A0)       ;Byte ausgeben
                MOVE.L  (A7)+,A0        ;A0 wiederherstellen
                RTS
SERAIN          MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  #$402000,A0     ;Adresse Port A
                bra     zda
SERBIN          MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  #$402010,A0     ;Adresse Port B
ZDA             BTST    #0,$3(A0)       ;Byte angekommen?
                BEQ     ZDA             ;nein
                MOVE.B  $7(A0),D0       ;Byte lesen
                MOVE.L  (A7)+,A0        ;A0 wiederherstellen
                RTS

SERASTAT        MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  $402003,A0      ;Statusregister Port A
SERSTAT         CLR     D0              ;zunaechst loeschen
                BTST    #2,(A0)         ;Transmitter sendebereit?
                BEQ     SEREINTEST      ;nein
                BSET    #0,D0           ;sonst Bit 0 setzen
SEREINTEST      BTST    #0,(A0)         ;Byte angekommen?
                BEQ     SERNIXDA        ;nein
                BSET    #1,D0           ;sonst Bit 1 setzen
SERNIXDA        MOVE.L  (A7)+,A0        ;A0 wiederherstellen
                RTS

SERBSTAT        MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  $402003,A0      ;Statusregister Port B
                BRA     SERSTAT         ;bearbeiten
PARBINIT        RTS
PARAINIT        MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  #$400000,A0     ;Grundadresse 68230
                MOVE.B  #$31,$1(A0)     ;MODE 0, H1 LOW-HIGH FLANKE
                MOVE.B  #$B1,$D(A0)     ;SUBMODE 10
                MOVE.B  #$FF,$5(A0)     ;ALLES AUSGAENGE
                MOVE.B  #$D,D0          ;CR SENDEN
                BRA     PARAS1
PARBOUT         RTS
PARAOUT         MOVE.L  A0,-(A7)        ;A0 retten
                MOVE.L  #$400000,A0     ;Grundadresse 68230
PARAW           BTST    #0,$1B(A0)      ;ACKN ANGEKOMMEN?
                BEQ     PARAW           ;nein
PARAS1          BSET    #0,$1B(A0)      ;ACKN-FLAG LOESCHEN
                MOVE.B  D0,$11(A0)      ;BYTE AUF LEITUNG SETZEN
                MOVE.B  #$B9,$D(A0)     ;STROBE ERZEUGEN
                move.l  d0,-(a7)        ;kurze Pause erzeugen
                move.l  (a7)+,d0        ;Strobelaenge = 39 Taktzyklen
                MOVE.B  #$B1,$D(A0)     ;STROBE LOESCHEN
                MOVE.L  (A7)+,A0        ;A0 wiederherstellen
                RTS
PARASTAT        MOVE.B  $40001B,D0      ;Statusregister lesen
                ANDI.B  #1,D0           ;Bit 0 gesetzt: Sendebereit
```

```
                                                ;Bit 1 gesetzt: Empfangsbereit = 0
                RTS
PARAIN          RTS                             ;entfaellt
PARBIN          RTS                             ;entfaellt
PARBSTAT        RTS
ADIn            TST.B       $C06000             ;AD-Wandlung starten
                MOVE        #10,D0              ;120 Zyklen warten
ADwarten        DBF         D0,ADwarten
                MOVE.B      $C04000,D0          ;Messwert lesen
                RTS
DAOut           MOVE.B      D0,$C02000          ;Byte in D0 ausgeben
                RTS

                ;Initport B initialisiert fuer die 8-Bit
                ;Parallelschnittstelle zu parallel angeschlossenen
                ;Host-Computern
                ;Bit0-3: Eingang
                ;Bit4-7: Ausgang
                ;Handshakeleitungen unbenutzt
InitPortB       move.l      A0,-(A7)            ;A0 retten
                lea         $400000,A0          ;Grundadresse 682J0
                move.b      #$35,$1(a0)         ;Mode 0, H12,H34 aktivieren,
                                                ;H1 und H3 reagieren auf H-L Fl
                                                ;im 2*4Bit Mode werden H3,H4
                                                ;nicht benoetigt
                move.b      #$80,$F(a0)         ;PortB Submode 10
                move.b      #$F0,$7(a0)         ;Bit0-3 = Eingang, 4-7 = Ausgang
                clr.b       $13(a0)             ;Ausgang loeschen
                move.l      (a7)+,A0            ;A0 wiederherstellen
                rts

ReadPortB       movem.l     A0/D1-D2,-(A7)      ;Register retten
                lea         $400013,A0          ;Adresse von Ausgabeport B
                clr         d0                  ;Einlesebyte zunaechst leer
                moveq       #2,d1               ;zuerst 0-3, dann 4-7 lesen
                move.b      #$80,(a0)           ;empfangsbereit
readBHS         move.b      (a0),D2             ;Lies den Eingang
                andi.b      #$C,d2              ;warte, bis korrektes Signal
                cmpi.b      #$4,d2              ;richtig?
                bne         readBHS             ;nein
Read01          btst        #2,$04(a0)          ;High Takt zurueck?
                beq         Read01              ;nein
                move.b      $4(a0),d2           ;sonst Daten lesen
                move.b      #$C0,(a0)           ;H-Takt ausgeben
                andi.b      #$3,D2              ;2 Bits isolieren
                or.b        d2,d0               ;einbauen
                ror.b #2,d0
Read23          btst        #2,$04(a0)          ;Low-Takt?
                bne         Read23              ;nein
                move.b      $04(a0),d2          ;Daten lesen
                andi.b      #$3,d2              ;2 Bits isolieren
                or          d2,d0               ;einbauen
```

```
                ror.b     #2,d0
                move.b    #$80,(a0)       ;L-Takt ausgeben
                subq      #1,d1           ;jetzt 4-7 lesen
                bne       read01          ;Schleife zu 1-mal wiederholen
                clr.b     (a0)            ;nicht empfangsbereit
                movem.l   (a7)+,A0/D1-D2  ;Register wiederherstellen
                rts

SendPortB       movem.l   A0/D0-D3,-(A7)  ;Register retten
                lea       $400013,A0      ;Port B des 68230
                clr.b     (a0)            ;nicht empfangsbereit
sendBHS         move.b    (a0),D2         ;Lies den Eingang
                andi.b    #$F,d2          ;nur Bits 0-3 interessieren
                cmpi.b    #$8,d2          ;empfangsbereit?
                bne       sendBHS         ;nein
                moveq     #2,d1           ;zuerst 0-3, dann 4-7 senden
                ror.b     #2,d0           ;richtige Position erzeugen
sendBw          ror.b     #2,d0           ;schieben
                move.b    d0,d2
                andi.b    #$30,d2         ;2 Bits isolieren
                ori.b     #$40,d2         ;H-Takt
                move.b    d2,(a0)         ;auf Port legen
send01          btst      #2,$4(a0)       ;Takteingang auf H?
                beq       send01          ;nein
                ror.b     #2,d0           ;jetzt 2-3 senden
                move.b    d0,d2
                andi.b    #$30,d2         ;Taktausgang auf L
                move.b    d2,(a0)         ;ausgeben
send23          btst      #2,$4(a0)       ;warten, bis Takteingang auf L
                bne       send23
                subq      #1,d1
                bne       sendBw          ;4 Paare senden
                clr.b     (a0)            ;nicht empfangsbereit
                movem.l   (a7)+,A0/D0-D3  ;Register wiederherstellen
                rts
```

```
**************************************************
*                                                *
*      1. Nachtrag zum C'T KAT-Ce Handbuch       *
*                                                *
**************************************************
```

Fehler im Handbuch:

S. 2: minimale AD-Wandlerzeit: ca 12 Mikrosek bei 8 MHz.
Dieser Wert ist nur bei direkter Ansteuerung erreichbar,
nicht aber durch den Aufruf der entsprechenden Sprungadresse
S. 22: unten, falsche Schreibweise, korrekt: ORG $1000,#4:PR132.CODE
S. 32: Sonderbearbeitung ist CHAR 11
Keypressbearbeitung ist CHAR 12
Tastaturbearbeitung ist CHAR 13
S. 53/54: Die Ausnahmevektoren sind nicht korrekt angegeben.

Ferner fehlt im Schaltplan der Hinweis auf die Leuchtdiode und den dazugehörigen Widerstand zwischen dem 68000 und den Eproms. Diese Bauteile sind in Reihe an GND und +5V geschaltet, um das Vorhandensein der Betriebsspannung anzuzeigen. Als Widerstand ist 390 Ohm zu wählen. Die Farbe der Leuchtdiode sollte auf jeden Fall mit der Farbe des Lötstopplackes harmonieren.

Ausnahmevektoren des 68000:
==========

Im untersten Adreßbereich des 68000 Prozessors befinden sich Zeiger, die im Monitorlisting nicht gut dokumentiert sind. Da diese Zeiger im RAM liegen, können diese Vektoren selbst verändert werden. Lediglich die beiden ersten Vektoren entfallen, da nach einem RESET-Impuls zunächst das Eprom in diesem Bereich liegt.

00: Stackpointer nach RESET
04: Startvektor nach RESET
08: Busfehler (nicht verfügbar)
0C: Adreßfehler
10: unbekannter Befehl
14: Division durch 0
18: CHK-Befehl
20: Privilegierte Verletzung
24: Trace
28: unbekannter Befehl der Form Axxx
2C: unbekannter Befehl der Form Fxxx
30..5F: reserviert (nicht benutzbar)
60: unechter Interrupt (nicht benutzbar)
64: Auto-Interrupt Level 1
68: Auto-Interrupt Level 2
6C: Auto-Interrupt Level 3
70: Auto-Interrupt Level 4
74: Auto-Interrupt Level 5
78: Auto-Interrupt Level 6
7C: Auto-Interrupt Level 7

80: Trap #0 (Trace ausschalten)
84: Trap #1 (Trace einschalten)
88: Trap #2
8C: Trap #3
90: Trap #4
94: Trap #5
98: Trap #6
9C: Trap #7
A0: Trap #8
A4: Trap #9
A8: Trap #10
AC: Trap #11
B0: Trap #12
B4: Trap #13
B8: Trap #14
BC: Trap #15
C0..FF: reserviert (entfällt)
ab 100: Monitorprogramm

Wie bereits im Handbuch angesprochen, wird der Inhalt des Eproms von Zeit zu Zeit überarbeitet, Fehler beseitigt (neue eingebaut?) sowie weitere Routinen implementiert.
Selbstverständlich können wir auch für neue Programmteile keine Funktionsgarantie übernehmen.

Reelle Zahlen für die C'T KAT-Ce

Als erste Erweiterung ist ab Version 1.4 die Verarbeitung reeller Zahlen erfolgt. Diese Zahlen werden intern in 2 Longwords (64 Bit) verarbeitet, wobei die Genauigkeit der Speicherung 47 Binaerstellen entspricht. Ausgegeben werden jeweils 13 Stellen in Dezimalschreibweise. Der Rechenbereich geht von 10 hoch -998 bis 10 hoch +998, dürfte also für die wesentlichen Anwendungen ausreichen, selbst wenn Sie Ihr GIRO-Konto verwalten wollen.

Den reellen Zahlen liegt eine duale Exponentialschreibweise in der Form:
VZ * 1.xxxxxxx.. mal 2 hoch EXP zugrunde.

Diese Darstellung ist folgendermaßen kodiert:

1. Longword: V**EEEEEEEEEEEEENDDDDDDDDDDDDDDD
2. Longword: DDDDCDDDDDDDDDDDDDDDDDDDDDDDDDDD

Bedeutung: V = Vorzeichen 1: Negativ 0: positiv
* = nicht benutzt: 0
E = Exponent
N = Nullhinweis N=1: Wert ungleich 0; N=0: reelle Zahl 0
D = Binaerwerte hinter dem Komma

Der dezimale Wert 1.0 ist kodiert als: 10018000_00000000
Der Exponent hat also die Bit-Kombination xxx1000000000001

Wird die Zweierpotenz größer, so wird für jeden Schritt der Exponent vergrößert. Damit hat die dezimale Zahl 16 die Darstellung: 10058000_0000000

Die dezimale Zahl -256 wäre demach: 90098000_00000000 (V2-Bit beachten)

Werden die Zahlen kleiner als 1, so wird der Exponent erniedrigt.

0.5 (dezimal) ist also: 10008000_00000000
und -0.0625 (dezimal): 8FFD8000_00000000

Nun kann z.B die Zahl 27 nicht als 2-er Potenz geschrieben werden. In 27 steckt als höchstmögliche Potenz 2 hoch 4 (16). Damit ist der Exponentialtei 1005. Jetzt wird der Rest sukzessive durch die Zweierpotenzen geteilt, die der Zahl 16 folgen und die oben als 'D' bezeichneten Bits von links nach rechts entsprechend dem Divisionergebnis gesetzt oder gelöscht.
27 durch 16 ist 1 Rest 11. Hier wird der Exponent verarbeitet.
11 durch 8 ist 1 Rest 3: höchstes D-Bit setzen
3 durch 4 ist 0 Rest 3: D-Bit auf 0
3 durch 2 ist 1 Rest 1: D-Bit setzen
1 durch 1 ist 1 Rest 0: D-Bit setzen
Da kein Rest bleibt, werden natürlich alle folgenden D-Bits gelöscht.
Damit ist die Darstellung von dezimal 27: 1005D800_00000000

Wählen wir eine Dezimalzahl: 8.0625
Exponentialwert: 3 -> 1004.....
8.0625 durch 8 ist 1 Rest 0.625
0.0625 durch 4 ist 0 Rest 0.0625: D=0
0.0625 durch 2 ist 0 Rest 0.0625: D=0
0.0625 durch 1 ist 0 Rest 0.0625: D=0
0.0625 durch 0.5 ist 0 Rest 0.0625: D=0
0.0625 durch 0.25 ist 0 Rest 0.0625: D=0
0.0625 durch 0.125 ist 0 Rest 0.0625: D=0
0.0625 durch 0.0625 Rest ist 1 Rest 0: D=1
alle anderen D-Bits auf 0 setzen.
also ist 8.0625: 10048100_00000000

```
*********************************************
*       2. Nachtrag zum KAT-Ce Handbuch     *
*                                           *
*          Eproms KAT-Ce Version 1.5        *
*          -------------------------        *
*********************************************
```

Zunächst einmal ist uns aufgefallen, daß eine wichtige Monitor-anweisung nicht im Handbuch abgedruckt ist, das Setzen von Register-inhalten. Es geht folgendermaßen:

R=D0:1234 oder R=A4:800128 oder r=I:5 usw.

Anderungen gegenüber 1.4:

ADIn und DAOut funktionieren

Die Routinen SerAStat und SerBStat funktionieren

DS funktioniert auch für Bereiche über 256 Byte

ADD xx,Ax funktioniert

Der Chain-Befehl funktioniert auch bei langen Files

EORI #xx,CCR wird korrekt übersetzt

BTST #xx,Dx wird auch für xx > 31 korrekt disassembliert

Indirekt indizierte Adressierung funktioniert auch mit negativem Offset

Die EQU Anweisung funktioniert auch dann, wenn vor dem Programmbeginn eine Konstante der Länge WORD definiert wird, deren Wert sich auf eine Speicherstelle bezieht, die hinter dem Programm steht.

Neu:

Die KAT-Ce erkennt ihre Systemfrequenz. Da hierzu der 68681 benutzt wird, muß dieser Baustein vorhanden sein, auch wenn die parallele Anschlußart angewendet wird. Die Frequenz wird auf 1000 Herz genau angegeben, wobei eine Abweichung von +/- 2000 Hz möglich ist. Für eigene Verwendung steht die Frequenz an der Adresse $3CC..$3CF als Longword.

Der gesamte Monitor wurde auf relative Adressierung geändert, weil damit leichter Spezialanpassungen möglich sind. Daher sehen die einzelnen Routinen anders als im Handbuchlisting aus (aus MOVEA.L #Adresse,A0 wurde z.B. LEA Adresse(PC),A0), die Logik sowie die Reihenfolge der Monitorprogramm-teile ist allerdings im wesentlichen unverändert.

Die KAT-Ce findet auch größere Ram-Konfigurationen, die ab $000000 bis $3FFFFF eingesetzt werden können (z.B durch externe Erweiterungen oder 4 weitere IC's 43256, die bei entsprechender Dekodierung auf die bisherigen RAM's aufgelötet werden können.

Der Ausnahmevektor bei $3C wird von der KAT-Ce nicht benötigt, er wurde für eine Sonderaufgabe vorgesehen. Wenn das dort stehende Longword von FFFFFFFF verschieden ist, wird das Programm ausgeführt, auf welches der Speicherinhalt zeigt. Falls das höchste Bit gesetzt ist, wird das Programm sofort ausgeführt, nachdem die Vektortabelle von $800000 nach $0 kopiert wurde, ist das höchste Bit nicht gesetzt, wo beginnt die Ausführung nach der Aufnahme der Verbindung zum Hostrechner. Das Programm wird als Unterprogramm aufgerufen, es ist also möglich, später in den normalen Betrieb überzugehen, wenn die Routine durch RTS abgeschlossen wird.

Im Hauptmenue wird die Option P(ascal angegeben, die allerdings nur dann ausgeführt wird, wenn ab $808008 ein Compiler zu finden ist. überprüft wird, ob das erste Wort ist $808008: 6000 (Branch Long). Sollten Sie einen eigenen Compiler haben, lassen Sie es uns nur wissen, ansonsten kann dort der KAT-Ce PASCAL-Compiler untergebracht werden.

Aus ebendiesem Grund ist der Ausgang des Editors geändert, es kann nur noch von dem Hauptmenue aus assembliert werden.

Dafür gibt es etwas neues. Sie können den Editor über $8001D0 (besser mit: MOVE #$34,-(A7) TRAP #4 siehe unten) aufrufen!. A0 muß einen Zeiger auf den Speicherbereich enthalten, ab dem editiert werden kann, D0 die Länge des Speicherbereiches. Wenn Ihnen jetzt die Idee kommt, damit ist es denkbar, von PASCAL aus den Editor als Prozedur aufzurufen, so liegen Sie völlig richtig.

Wo wir schon von PASCAL schreiben. Für ein Multitasking (bei uns wird es ein sehr einfaches sein) wäre eine interruptgesteuerte Unterbrechung sinnvoll. Haben wir schon eingebaut: MOVE #$35,-(A7) und TRAP #4 starten oder stoppen diesen Interrupt. In D0 steht die Anzahl der Millisekunden, die zwischen 2 Interruptimpulsen vergehen soll, also Taktzeit 10 Millisekunden: MOVEQ #10,D0. Soll der Interrupt gesperrt werden, so löschen Sie D0 vor dem Systemtrap. Das Register A0 sollte die Adresse der Interruptroutine beinhalten, wo Sie den Interrupt abarbeiten wollen. Beachten Sie, daß der Befehl: MOVE.B #1,$400035 dort vorkommt, damit der Interrupt beendet werden kann. Daß eine Interruptroutine mit RTE abzuschließen ist, wissen Sie ja wohl sowieso.
Wollen Sie keine eigene Interruptroutine schreiben, so löschen Sie das A0-Register vor Aufruf der Routine. In diesem Fall wird unsere eingebaute Interruptroutine ausgewählt. Diese löscht zunächst die Speicherstelle $42C, bei jedem Interrupt wird sie hochgezählt.
Soll der Interrupt in kritischen Situationen gesperrt werden, so verwenden Sie den Befehl ORI #$0700,SR
Sobald wieder interruptiert werden darf, folgt: ANDI #$F8FF,SR

Damit der Interrupt überhaupt vom 68230 an die CPU weitergeleitet wrid, muß unbedingt auf Position 3 des Interruptjumperfeldes eine Brücke gesetzt werden!

An der Speicherstelle $3D0 steht ein Zufallswert (Long), welcher durch die Kommunikationsroutinen mit dem Hostrechner verändert wird.

Während eines Trace-Laufes kann durch Drücken der '?' Taste der Monitor aufgerufen werden. Sie sollten unbedingt nur Speicherstellen ansehen oder verändern, keinesfalls aber Routinen neu starten!

Die Systemroutinen (z.B Keypress, Byteout usw) sind nun über TRAP #4 erreichbar, wobei vor dem Trapaufruf die Nummer der gewünschten Operation auf den Stack gegeben werden muß. Dabei leiten sich diese Nummern direkt aus der Reihenfolge in der unverändert existierenden Sprungtabelle ab $800100 ab. Die Register werden nur noch in den angegebenen Fällen verändert. Nach dem Trap #4 Aufruf wird normal weitergearbeitet. Sollte das Tracen eingestellt sein, so werden die Monitorroutinen 'im verborgenen' ausgeführt, anschließend wird weitergetraced. Wenn Sie wollen, daß Ihre Routinen später auch auf Nachfolgern der KAT-Ce oder anderen Systemen, auf die wir (vielleicht) unsere Software implementieren, laufen sollen, so sollten Sie ab sofort nur noch mit TRAP #4 arbeiten!

```
$00: SetResetAdr    ;ResetAdresse in A0 nach RESET setzen
$01: PreInit        ;Kaltstart
$02: MonInit        ;Monitor als Unterprogram aufrufen, alle Register ver.
$03: Quit           ;Ende der Verbindung zum Hostrechner
$04: Keypress       ;D0 ist 1, wenn Zeichen anliegt
$05: ByteIn         ;Zeichen in D0
$06: TastIn         ;Zeichen in D0
$07: GetString      ;A1 zeigt auf String, Länge in (A1)
$08: ByteOut        ;D0 ausgeben
$09: AsciOut        ;D0 als Text ausgeben
$0A: WordOut        ;D0 als Wort ausgeben
$0B: PrtByt         ;D1 hex ausgeben
$0C: PrtWord        ;D1 asl Hexwort ausgeben
$0D: StrOut         ;String in (A1) ausgeben, A1 ver.
$0E: CROut          ;Zeilenrücklauf und Vorschub erzeugen, D0 ver.
$0F: ClBild         ;Bild löschen, D0 ver.
$10: PrtCol         ;Doppelpunkt ausgeben, D0 ver.
$11: PrtMSpc        ;(D1)+1 Leerzeichen ausgeben, D0-D1 ver.
$12: GotoXY         ;D1.W enthält im Lowbyte x, im Highbyte y-Koord.
$13: Bell           ;Piep ausgeben, D0 ver.
$14: DisZeile       ;Adresse in A1, nachher in A1 neue Adresse,
                    ;A0-A4/D0-D4 ver.
$15: ResetFile      ;A1 zeigt auf Namenstring, D1 Filenummer
$16: RewriteFile    ;A1 zeigt auf Namenstring, D1 Filenummer
$17: BlockRead      ;Filenummer in D1, Blocknummer in D5
$18: BlockWrite     ;Filenummer in D1, Blocknummer in D5
$19: CloseFile      ;Filenummer in D1
$1A: CloseDeleteFile     ;Filenummer in D1
$1B: IOResult       ;Ergebnis in D0
$1C: UnitOut        ;D0 auf Hostdrucker ausgeben
$1D: UnitInit       ;Hostdrucker initialisieren
$1E: DAOut          ;D0 mit ZN 428 ausgeben
$1F: ADIn           ;ZN 427 in D0 einlesen
$20: ParAInit       ;Centronicsschnittstelle initialisieren
$21: ParAStat       ;Status nach D0 einlesen
$22: ParAOut        ;D0 an Drucker ausgeben
$23: ParAIn         ----
$24: ParBInit       ----
$25: ParBStat       ----
$26: ParBOut        ----
$27: ParBIn         ----
$28: SerAInit       ;Init Seriell Port A (kleine Schnittstelle)
                    ;D1 enthält Baudratendefintion
                    ;immer 8 Bit, 1 Stoppbit, keine Parität
$29: SerAStat       ;Bit 0 von D0 gesetzt, wenn sendebereit
                    ;Bit 1 von D0 gesetzt, wenn Byte angekommen
$2A: SerAOut        ;D0 ausgeben
$2B: SerAIn         ;D0 einlesen
$2C: SerBInit       ;Init Seriell Port B (große Schnittstelle)
                    ;D1 enthält Baudratendefintion
                    ;immer 8 Bit, 1 Stoppbit, keine Parität
$2D: SerBStat       ;Bit 0 von D0 gesetzt, wenn sendebereit
                    ;Bit 1 von D0 gesetzt, wenn Byte angekommen
```

```
      $2E: SerBOut        ;D0 ausgeben
      $2F: SerBIn         ;D0 einlesen
      $30: InitPortB      ;parallele Schnittstelle zum Hostrechner init.
      $31: ReadPortB      ;D0 lesen
      $32: SendPortB      ;D0 ausgeben
      $33: Real           ;D7 enthält Art der Real-Operation

Neu:  $34: Editor         ;A0 zeigt auf Startadresse des Speicherbereiches,
                          ;in dem editiert werden darf, D0 enthält die
                          ;Länge des reservierten Speicherplatzes.
                          ;Alle Register können verändert sein!

      $35: Interrupt      ;D0 = 0 schaltet Interrupt ab
                          ;D0 <> 0 gibt an, wieviel Millisek. Taktzeit
                          ;$42C zählt im Interrupttakt hoch
                          ;falls A0 = 0, wird die interne Interruptroutine
                          ;gewählt, ist A0 <> 0, so wird die in A0 stehende
                          ;Adresse in die Interruptvektortabelle bei $68
                          ;eingetragen
```

Trap #4 - Beispiel:

```
          lea     string,A1
          move    #$0D,-(A7)    ;Anwählen von STROUT
          trap    #4
          rts
String    str     'String durch TRAP #4 ausgegeben'
```

Selbstverständlich ist es nicht möglich, das Handbuch immer dann neu zu drucken, wenn eine bessere (?) Epromversion zur Verfügung steht. Aus diesem Grund bieten wir den aktuellen Monitor-Source-Text (also in demselben Umfang wie im Handbuch beschrieben, ohne Disassembler, Real, Editor, Assembler) auf folgenden Diskettenformaten an:

APPLE II Prodos (143k-Laufwerke), ATARI ST, IBM PC, Gepard (3 1/2 Zoll).

Ansonsten können wir ein Listing ausdrucken.
Der Preis für das Listing oder die Diskette beträgt jeweils 40 DM.

Läßt sich eine Zahl nicht als Summe von Zweierpotenzen schreiben,
so tritt in der REAL-Darstellung eine Periode auf, ein Beisp. entfällt.

Demonstration einer Umwandlung REAL in Dezimal:
        100D9340_80000000
Es fällt zunächst auf, daß die Zahl positiv ist.
Die Zweierpotenz ist 12, also Grundwert 4096.
Die Bitfolge der D-Bits ist: x001.0011.0100.0000 1000.0000....
Das erste wertsteigernde Bit ist das 3., also wird 2 hoch 9 addiert usw.
Ergebnis: 2^12 + 2^9 + 2^6 + 2^5 + 2^3 + 2^-4
       = 4096 + 512 + 64 + 32 + 8   + 0.0625
       = 4712.0625   (dezimal)

Aufruf der Operationen für reelle Zahlen:
==
Die Real-Operationen sind über eine neue Sprungadresse erreichbar:
$8001CC (siehe dazu vergleichbare Sprünge auf S. 17 des Handbuches).
Die gewünschte Operation wird durch den Inhalt des Datenregisters D7
ausgewählt. Nach der Operation gibt der Inhalt von D7 eine Fehlermeldung,
sofern D7 von 0 verschieden ist (Ausnahme: Vergleiche).
Die Realwerte werden in den Registerpaaren D0_D1 bzw. D2_D3 übergeben,
bei Stringoperationen zeigt A1 auf den String. Außer D0_D1 und D7 werden
keine Register verändert.

Liste der durch D7 erreichbaren Operationen:

```
                Stringoperationen:
                ==================
0: Umwandlung String zu Real  String in (A1), Real in D0_D1
                              im Highword von D7 steht, wieweit diese
                              Routine den String abgearbeitet hat.
1: Umwandlung Real zu String  D0_D1 -> (A1) Exponentialformat
                              D2 = minimale Länge des Strings, vorne werden
                                   gegebenenfalls Leerzeichen eingefügt
                              D3 = Nachkommastellen
2: Umwandlung Real zu String  D0_D1 -> (A1) Festkommaformat, D2, D3 wie oben
3: Umwandlung Real zu String  D0_D1 -> (A1) Ingenieur-Format, D2, D3 wie oben
4: Umwandlung Real zu String  D0_D1 -> (A1) Gleitkommaformat, D2, D3 wie oben
                Vergleiche:
                ===========
5: gleich               D0_D1 =  D2_D3       Vergleiche: D7 = 1, wenn wahr
6: ungleich             D0_D1 <> D2_D3                   D7 = 0, wenn falsch
7: kleiner oder gleich  D0_D1 <= D2_D3
8: kleiner              D0_D1 <  D2_D3
9: groesser oder gleich D0_D1 >= D2_D3
10:groesser             D0_D1 >  D2_D3
                Grundrechenarten:
                =================
11: Vorzeichen kippen       - D0_D1 -> D0_D1
12: Multiplizieren    D0_D1 * D2_D3 -> D0_D1
13: Dividieren        D0_D1 / D2_D3 -> D0_D1
14: Addieren          D0_D1 + D2_D3 -> D0_D1
15: Subtrahieren      D0_D1 - D2_D3 -> D0_D1
16: Quadrieren        D0_D1 * D0_D1 -> D0_D1
17: Kehrwert bilden     1 / (D0_D1) -> D0_D1
                Real-Integer Umrechnungen:
                ==========================
18: Real zu Longinteger  D0_D1 -> D0
19: Longinteger zu Real  D0    -> D0_D1
```

```
                Transzendente Funktionen:
                =========================
20: Sinus              sin (D0_D1) -> D0_D1
21: Cosinus            cos (D0_D1) -> D0_D1
22: Tangens            tan (D0_D1) -> D0_D1
23: Arcustangens   arctan (D0_D1) -> D0_D1
24: Arcussinus     arcsin (D0_D1) -> D0_D1
25: Arcuscosinus   arccos (D0_D1) -> D0_D1
26: Wurzel ziehen    SQRT (D0_D1) -> D0_D1
27: Zweierpotenz   2 hoch (D0_D1) -> D0_D1
28: Exponential    e hoch (D0_D1) -> D0_D1
29: Zehnerpotenz  10 hoch (D0_D1) -> D0_D1
30: Potenz   (D0_D1) hoch (D2_D3) -> D0_D1
31: Zweierlog.          lb (D0_D1) -> D0_D1
32: nat. Log.           ln (D0_D1) -> D0_D1
33: Zehnerlog.         log(D0_D1) -> D0_D1
                Sonstiges:
                ==========
34: Pi-Erzeugen                Pi -> D0_D1
35: Fakultaet            (D0_D1)! -> D0_D1
36: Gauss-Klammer von D0_D1 (Nachkommastellen abschneiden)
37: Vorkommastellen von D0_D1 beseitigen

Fehler in D7 : D7 =0, wenn kein Fehler
               Bit 0 gesetzt, wenn Ueberlauf erzeugt
               Bit 1 gesetzt, wenn Division durch 0
               Bit 2 gesetzt, wenn Wurzel oder Log. aus neg Zahl
               Bit 3 gesetzt, wenn ArcSin nicht im Bereich -1..+1 liegt
               Bit 4 gesetzt, wenn Syntaxfehler bei String zu Real
               Bit 5 gesetzt, wenn unbekannte Real-Operation

Beispiel für eine Realrechnung:

Getstring  equ  $80011C
Piep       equ  $80014C
Real       equ  $8001CC
Stringout  equ  $800134

start   lea     string,A1
        jsr     Getstring
        moveq   #0,D7          ;String zu Real
        jsr     Real
        moveq   #32,D7         ;nat. Logar.
        jsr     real
        tst     D7             ;Fehler?
        bne     fehler         ;ja ->
        lea     string,A1
        moveq   #1,D7          ;Real zu String
        jsr     Real
        jsr     Stringout
        rts
fehler  jmp     Piep
string  ds      40
```